J Comput Virol (2010) 6:1–29
DOI 10.1007/s11416-009-0127-3

ORIGINAL PAPER

Applied parallel coordinates for logs and network traffic attack analysis

Sebastien Tricaud · Philippe Saadé

Received: 20 December 2008 / Accepted: 17 July 2009 / Published online: 27 August 2009

Abstract By looking on how computer security issues are handled today, dealing with numerous and unknown events is not easy. Events need to be normalized, abnormal behaviors must be described and known attacks are usually signatures. Parallel coordinates plot offers a new way to deal with such a vast amount of events and event types: instead of working with an alert system, an image is generated so that issues can be visualized. By simply looking at this image, one can see line patterns with particular color, thickness, frequency, or convergence behavior that gives evidence of subtle data correlation. This paper first starts with the mathematical theory needed to understand the power of such a system and later introduces the Picviz software which implements part of it. Picviz dissects acquired data into a graph description language to make a parallel coordinate picture of it. Its architecture and features are covered with examples of how it can be used to discover security related issues.

Keywords Visualization · Parallel coordinates · Data-mining · Logs · Computer security

A picture a day keeps the doctor away.

S. Tricaud
Honeynet Project French Chapter, 69 rue Rochechouart,
75009 Paris, France
e-mail: sebastien@honeynet.org

P. Saadé (✉)
Lycée la Martinière Monplaisir, Laboratoire de Mathématiques,
41, rue Antoine Lumière, 69372 Lyon Cedex 08, France
e-mail: psaade@gmail.com

1 Introduction

This paper covers how visualization techniques based on parallel coordinate plots (abbreviated as //-coords) can enhance the computer security area.

It is common to have thousands lines of logs a day on a single machine. With private networks of hundreds of computers over complex topologies, this really represents a huge load of information. How can one separate the important part of the information from the unimportant one?

To deal with that issue, administrators, most of the time, use tools such as Prelude LML,[1] OSSEC[2] or similar software that are often based on signatures. Besides signatures based tools, they also use anomaly based tools, that are classifying the information after a learning phase. One example is spamassassin,[3] which does a great job at removing spam out of our mailboxes. Over the years, these tools have proven an indisputable efficiency.

However, something missing today is dealing with data exactly as it is. There is often more to see than just the part of the data having a matching threshold of signature. That's why computer visualization is a good choice!

Computer visualization is a neat way to see the picture of what is really happening and can, in some cases, handle a lot of information. As //-coords can handle multiple dimensions and an infinity of events, it became a natural choice to write a software being able to automate those graphs creation. This software is called Picviz.

In the first part of this paper, we will introduce the very basic facts about //-coords. We will explain in the most simple terms the fundamentals of //-coords as a mathematical

[1] http://www.prelude-ids.org.
[2] http://www.ossec.net.
[3] http://spamassassin.apache.org.

theory. The first important results will be given, without assuming from the reader a heavy mathematical background.

In the second part, we will present Picviz in its overall architecture and then in greater detail.

The last part of this article will be devoted to real-life examples.

We will first discuss the case of giant log files of Cray systems, whose syntax was unknown to the authors and gave a challenging playground for finding important events without using any pre-existing signatures or tools of any kind.

Then, we will consider the particular case of Botnet attacks. It came out that //-coords and Picviz can help in detecting and characterizing these malicious threats. And we will explain how in the last section of this paper.

Examples including Apache access.log file and wireless LAN traffic monitoring will also be given.

2 A short mathematical introduction to parallel coordinates

2.1 Cartesian and parallel point of view

Imagine one has to collect elementary events of a given type (temperatures of all capitals of Asia, network traffic on a network adapter, etc.). Let's suppose that each given elementary event carries N kinds of information and that N is not small (greater than 4). Since it is not easy to plot vectors belonging to a space of more than 3 dimensions in a 3 dimensional physical space (not counting the time), it becomes necessary to adapt the representation technique.

In an N-dimensional vector space E, one needs a basis of N vectors. Then each vector $\vec{u} \in E$ corresponds to an N-tuple of the form $(x_1, x_2, \ldots, x_n)$. In the usual euclidean space of dimension N, denoted $\mathbb{R}^N$, the canonical basis is orthogonal, which means that axes are considered pairwise perpendicular (Fig. 1).

This is the usual cartesian representation of $\mathbb{R}^3$ and it corresponds to our everyday life experience of our ambient space! But since it is impossible to draw more than 3 perpendicular axes in a 3 dimensional physical space, the idea behind //-coords is to draw the axes side by side, all parallel

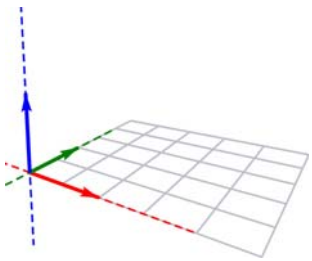


Fig. 1 Orthogonal basis in $\mathbb{R}^3$

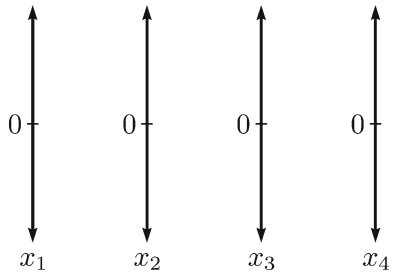


Fig. 2 Four axes

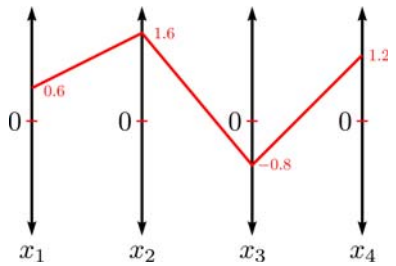


Fig. 3 Four axes and a vector

to a given direction. It is then possible to draw all these axes in a 2d plane (Fig. 2):

For example, the vector $\vec{u} = (0.6, 1.6, -0.8, 1.2) \in \mathbb{R}^4$ should show up as (Fig. 3).

That point of $\mathbb{R}^4$ has become a polygonal line in //-coords!

At first sight, it might seem that we have lost simplicity. Of course, on one side, it is obvious that many points will lead to many polygonal lines, overlapping each other in a very cumbersome manner. But on the other side, it is a fact that certain relationships between coordinates of the point correspond to interesting patterns in //-coords.

It is the aim of this short mathematical introduction to present some elementary patterns that can be observed and to prepare the reader for the definition of geometrical invariants (in //-coords) of simple subspaces of $\mathbb{R}^N$ such as lines, planes and p-subspaces (sometimes called p-flats).

2.2 Trivial hyperplanes

The usual definition of **an affine hyperplane** in $\mathbb{R}^N$ is that it is the set of vectors $\overrightarrow{u} = (x_1, x_2, \ldots, x_N)$ satisfying an affine relationship of the form

$$\alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_N x_N = \beta$$

where all coefficients $\alpha_1, \ldots, \alpha_N$ and β are real numbers and $\alpha_1, \ldots, \alpha_N$ are not zero all together.

That hyperplane is **a vector hyperplane** or simply **a hyperplane** if $\beta = 0$ which geometrically means that it passes through to origin $\mathcal{O} = (0, 0, \ldots, 0)$.

For example, in the usual plane $\mathbb{R}^2$, the relationship

$$\alpha_1 x_1 + \alpha_2 x_2 = \beta, \quad \text{where} \quad (\alpha_1, \alpha_2) \neq (0, 0)$$

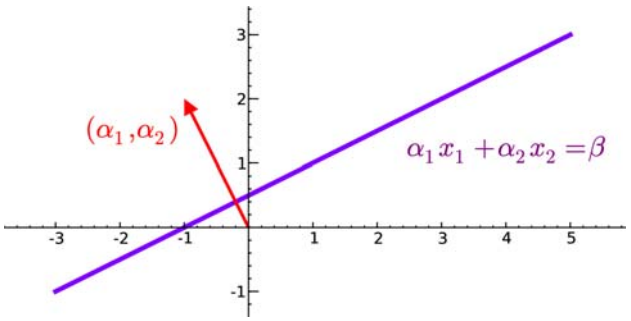


Fig. 4 A line in $\mathbb{R}^2$

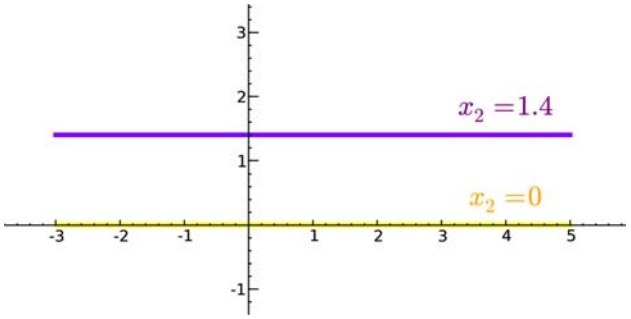


Fig. 5 A coordinate line and a trivial line in $\mathbb{R}^2$

characterizes a line orthogonal to vector $\overrightarrow{n} = (\alpha_1, \alpha_2)$ (Fig. 4).

Naturally, in $\mathbb{R}^3$, the cartesian equation

$$\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = \beta, \quad \text{where } (\alpha_1, \alpha_2, \alpha_3) \neq (0, 0, 0)$$

defines a plane having $\overrightarrow{n} = (\alpha_1, \alpha_2, \alpha_3)$ as normal vector.

Hyperplanes generalize these geometric objects in higher dimension and are precisely the $(N-1)$-affine subspaces of $\mathbb{R}^N$.

Trivial hyperplanes are those which can be defined by an equation of the form

$$x_i = b$$

for a fixed given index i.

In such a case, that hyperplane is simply parallel to the i**th coordinate hyperplane** having equation $x_i = 0$ (Fig. 5).

In //-coordinates, such trivial hyperplanes are very easy to recognize since one coordinate is constant (Fig. 6):

This is the first simple pattern to show up in //-coordinates. For example, if one studies a set of TCP/IP packets, one can assign to the first axis the source IP address and to the second axis the destination port. It is then obvious that in an attack such as DOS on port 80 (www) of a server coming from many different machines, such a structure is going to appear. And even with much more than two axes involved, if one of them represents destination port, the above pattern will still be there, and easy to notice.

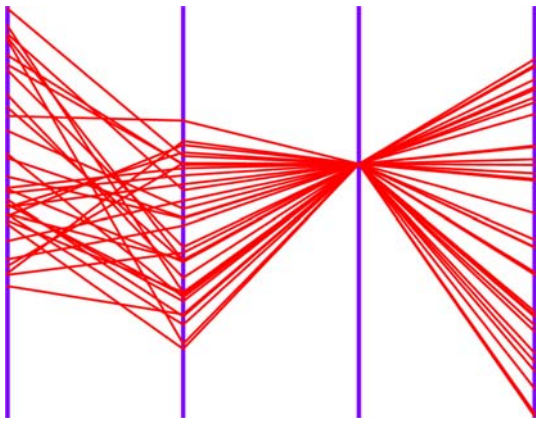


Fig. 6 A trivial hyperplane in $\mathbb{R}^4$

Now that we have seen that **trivial hyperplanes** are easy to detect in //-coordinates, we must consider **nontrivial** ones. Are they so easy to discover? For example, let's say you have 10000 points scattered on a fixed affine hyperplane $\mathcal{H}$ in $\mathbb{R}^5$. Will the //-coordinates plot of these 10000 points present a trivial pattern that can be seen at first glance? The definitive answer is *No!* or, more precisely, *Not yet!*.

So, do not despair! In the following section, we will try to explain in the most simple terms *what* can be seen and *how* it can be seen using //-coordinates.

3 A detailed overview of some basic facts of parallel coordinates

3.1 Notations and conventions

As said earlier, all //-coordinates plots are drawn on a 2-dimensional plane. To keep things simple, all the axes will be drawn vertically. The plane on which they are drawn is equipped with a usual cartesian coordinate system, denoted

$$\mathcal{R}_{/\!/} = (\mathcal{O}, \overrightarrow{\imath}, \overrightarrow{\jmath})$$

where $(\overrightarrow{\imath}, \overrightarrow{\jmath})$ is an orthonormal basis (Fig. 7).

A point M in that plane will have its coordinate relative to $\mathcal{R}_{/\!/}$ given by

$$M = \left(x_{/\!/}, y_{/\!/}\right)_{\mathcal{R}_{/\!/}}$$

We will denote by ε_i the abscissa of the vertical line carrying axis x_i in $\mathcal{R}_{/\!/}$

$$(x_i): \quad x_{/\!/} = \varepsilon_i$$

Definition $\mathcal{R}^N_{/\!/}(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N)$ corresponds to $\mathcal{R}_{/\!/}$ equipped with N vertical axes having abscissas $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_N$.

To shorten the notation, we will often simply write $\mathcal{R}^N_{/\!/}$.

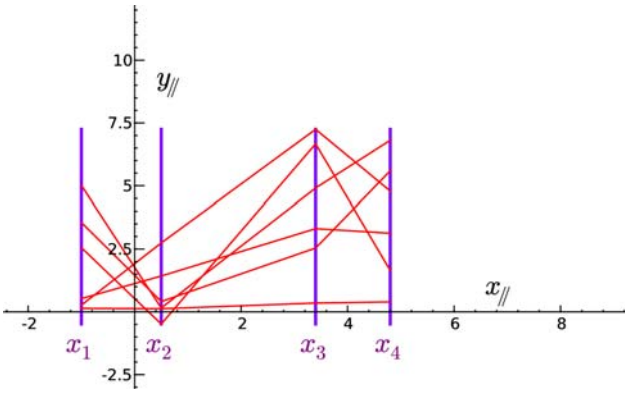


Fig. 7 A //-coordinate system for $\mathbb{R}^4$

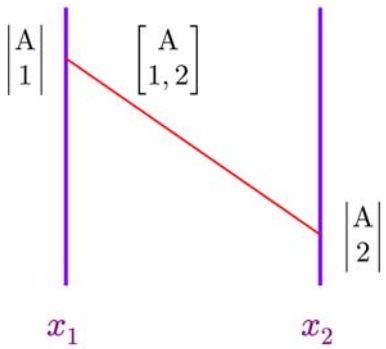


Fig. 8 Notations

And sometimes, we will even forget about the conditions $\varepsilon_1 < \varepsilon_2 < \cdots < \varepsilon_N$, just for fun!

Definition If $A = (a_1, a_2, \ldots, a_N) \in \mathbb{R}^N$, let $\left|\begin{matrix}\mathrm{A}\\ i\end{matrix}\right|$ denote the point

$$\boxed{\left|\begin{matrix}\mathrm{A}\\ i\end{matrix}\right|_{\mathcal{R}_{/\!/}} = (\varepsilon_i, a_i)_{\mathcal{R}_{/\!/}}}$$

This is simply the point attached to axis x_i and by which passes the polygonal line representing A (Fig. 8).

Definition The line joining $\left|\begin{matrix}\mathrm{A}\\ i\end{matrix}\right|$ and $\left|\begin{matrix}\mathrm{A}\\ i+1\end{matrix}\right|$ will be denoted by

$$\boxed{\left|\begin{matrix}\mathrm{A}\\ i, i+1\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left(\left|\begin{matrix}\mathrm{A}\\ i\end{matrix}\right| \left|\begin{matrix}\mathrm{A}\\ i+1\end{matrix}\right|\right)}$$

and, when needed, the segment joining these two points will be denoted by

$$\left[\begin{matrix}\mathrm{A}\\ i, i+1\end{matrix}\right]$$

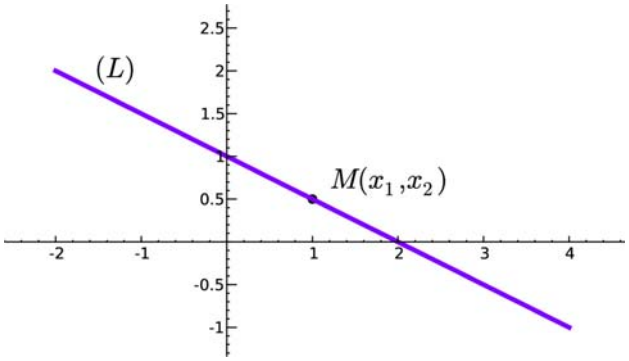


Fig. 9 (L) in $\mathbb{R}^2$

Remark Most of the time, we will not make any difference between the line $\left|\begin{matrix}\mathrm{A}\\ i, i+1\end{matrix}\right|$ and the segment $\left[\begin{matrix}\mathrm{A}\\ i, i+1\end{matrix}\right]$. For sake of simplicity, we will often draw segments and build intersection points of such segments as if they were lines.

Remark Sometimes, we will consider lines joining points on axes that are not consecutive. For example, $\left|\begin{matrix}\mathrm{A}\\ i, j\end{matrix}\right|$ corresponds to the line joining $\left|\begin{matrix}\mathrm{A}\\ i\end{matrix}\right|$ and $\left|\begin{matrix}\mathrm{A}\\ j\end{matrix}\right|$, even if $|i - j| \neq 1$. The general cartesian equation of such a line is

$$\boxed{\left|\begin{matrix}\mathrm{A}\\ i, j\end{matrix}\right|_{\mathcal{R}_{/\!/}} \quad : \quad (y_{/\!/} - a_i)(\varepsilon_j - \varepsilon_i) = (x_{/\!/} - \varepsilon_i)(a_j - a_i)}$$

or

$$\boxed{\left|\begin{matrix}\mathrm{A}\\ i, j\end{matrix}\right|_{\mathcal{R}_{/\!/}} \quad : \quad y_{/\!/} = a_i + (x_{/\!/} - \varepsilon_i)\frac{(a_j - a_i)}{(\varepsilon_j - \varepsilon_i)}}$$

3.2 A first glance at the case of a line $\mathcal{L}$ in $\mathbb{R}^2$

In this section, we are going to focus on a line $(\mathcal{L})$ of $\mathbb{R}^2$ (Fig. 9).

The way such a line looks like in //-coordinates mainly depends on the slope of $\mathcal{L}$.

To get an intuitive feeling of what is going on, have a look at the following plots of $\mathcal{L}$ in //-coords for different slopes (Fig. 10):

It seems obvious that in all cases, when $M(x_1, x_2)$ describes $\mathcal{L}$, the line $\left|\begin{matrix}\mathrm{M}\\ 1, 2\end{matrix}\right|$ passes by a fixed point who's horizontal position solely depends on the slope of $\mathcal{L}$.

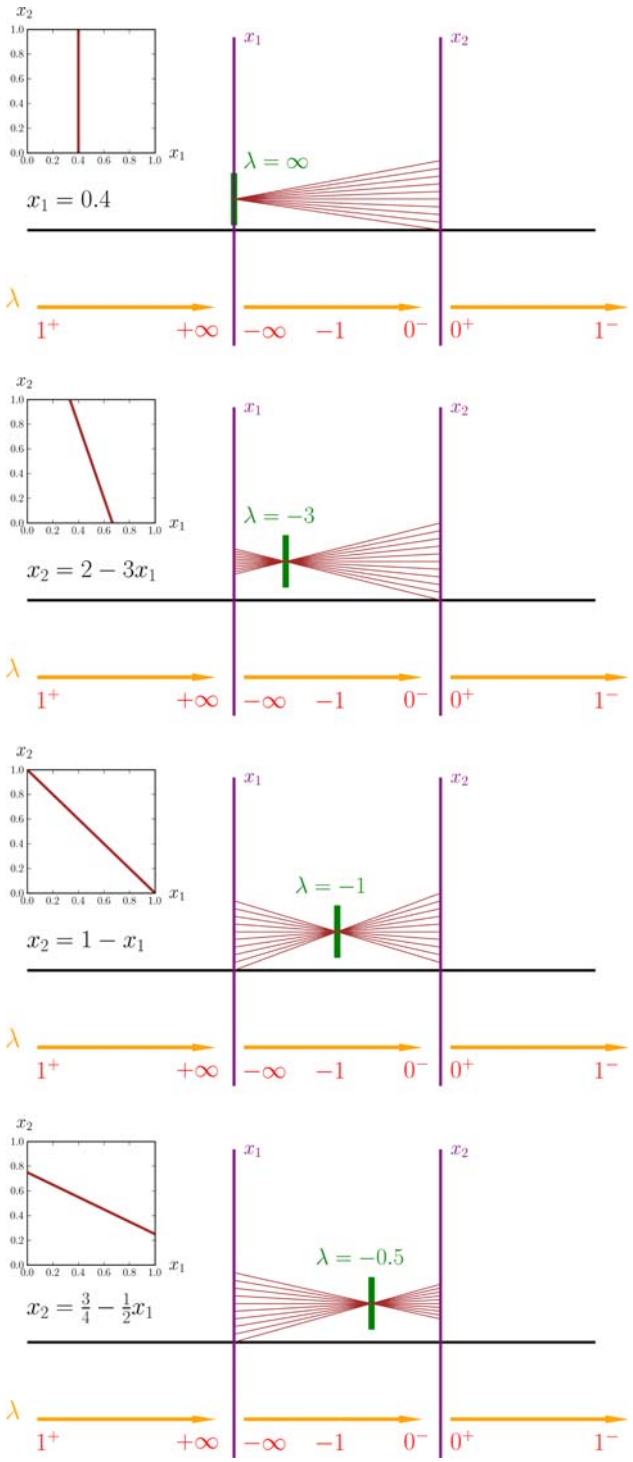


Fig. 10 Different slopes for (L) in $\mathbb{R}^2$

Fig. 10 continued

3.3 A closer look at the case of a line $\mathcal{L}$ in $\mathbb{R}^2$

Let $M(x_1, x_2) \in \mathcal{L}$. Then the corresponding line in $\mathcal{R}^2_{/\!/}$ has the following equation

$$\left| \begin{matrix} \mathrm{M} \\ 1, 2 \end{matrix} \right|_{\mathcal{R}_{/\!/}} \quad : \quad (y_{/\!/} - x_1)(\varepsilon_2 - \varepsilon_1) = (x_{/\!/} - \varepsilon_1)(x_2 - x_1)$$

- In the particular case where $\mathcal{L}$ is parallel to the first diagonal $\Delta : x_2 = x_1$, that is, when $\mathcal{L} : x_2 = x_1 + \beta$, one has (Fig. 11)

$$\left| \begin{matrix} \mathrm{M} \\ 1, 2 \end{matrix} \right| \quad : \quad y_{/\!/} = x_1 + (x_{/\!/} - \varepsilon_1)\frac{\beta}{(\varepsilon_2 - \varepsilon_1)}$$

Fig. 11 $(\mathcal{L}):\ x_2 = x_1 + \beta$

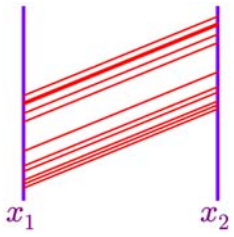
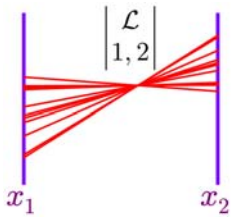


Fig. 12 Random points on a line $\mathcal{L}$ of $\mathbb{R}^2$



meaning that when M describes all of $\mathcal{L}$, the line $\left|\begin{matrix}\mathrm{M}\\1,2\end{matrix}\right|$ remains parallel to vector $\begin{pmatrix}1\\ \frac{\beta}{\varepsilon_2-\varepsilon_1}\end{pmatrix}_{\mathcal{R}_{/\!/}}$ or $\begin{pmatrix}\varepsilon_2-\varepsilon_1\\ \beta\end{pmatrix}_{\mathcal{R}_{/\!/}}$.

- In the more general case where $\mathcal{L}:\alpha_1x_1+\alpha_2x_2=\beta$ is not parallel to $\Delta: x_2 = x_1$, that is, when $\alpha_1+\alpha_2\neq 0$, it can be easily shown that all lines $\left|\begin{matrix}\mathrm{M}\\1,2\end{matrix}\right|$ have a common point (Fig. 12).

We will denote that point by $\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|$, and a straightforward computation gives

$$\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left(\frac{\alpha_1\varepsilon_1+\alpha_2\varepsilon_2}{\alpha_1+\alpha_2}, \frac{\beta}{\alpha_1+\alpha_2}\right)_{\mathcal{R}_{/\!/}}$$

Remark People used to working with projective spaces will immediately notice that both cases can be described by the point

$$\boxed{\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [\alpha_1\varepsilon_1+\alpha_2\varepsilon_2 : \beta : \alpha_1+\alpha_2]}$$

in $\mathbb{RP}^2$.

From now on, we will consider that any line $\mathcal{L}\subset\mathbb{R}^2$ corresponds to a point $\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|$ in $\mathcal{R}^2_{/\!/}$ and we will not mention anymore that in some cases that point has to be defined in projective plane.

Remark Reciprocally, given the point $\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = (x_{0/\!/}, y_{0/\!/})_{\mathcal{R}_{/\!/}}$,
one recovers a cartesian equation of $\mathcal{L}$

$$\mathcal{L}:\quad px_1+(1-p)x_2 = y_{0/\!/},\quad \text{where } p = \frac{x_{0/\!/}-\varepsilon_2}{\varepsilon_1-\varepsilon_2}$$

The correspondence between $\mathcal{L}$ and $\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|$ is called by Inselberg **the line-point duality**:

$$\boxed{\mathcal{L}:\ \alpha_1x_1+\alpha_2x_2=\beta}$$
$$\downarrow$$
$$\boxed{\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [\alpha_1\varepsilon_1+\alpha_2\varepsilon_2 : \beta : \alpha_1+\alpha_2]}$$

$$\boxed{\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [x_{0/\!/}, y_{0/\!/}, z_{0/\!/}]}$$
$$\downarrow$$
$$\boxed{\mathcal{L}:\ (x_{0/\!/}-\varepsilon_2z_{0/\!/})x_1+(\varepsilon_1z_{0/\!/}-x_{0/\!/})x_2 = y_{0/\!/}(\varepsilon_1-\varepsilon_2)}$$

3.4 Vocabulary

The preceding point $\left|\begin{matrix}\mathcal{L}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}}$ will be called **the level 2 invariant point** of line $\mathcal{L}$, or simply **the invariant point** of line $\mathcal{L}$, in parallel coordinates.

4 First nontrivial results in //-coordinates

4.1 A first glance at the case of a plane $\mathcal{P}$ in $\mathbb{R}^3$

As we did for the case of a line $\mathcal{L}$ in $\mathbb{R}^2$, we will start by some simple geometrical configuration of an affine plane $\mathcal{P}$ in $\mathbb{R}^3$.

First of all, we already know that if $\mathcal{P}$ is a trivial plane $x_i = \beta$ then random points on $\mathcal{P}$ show up in $\mathcal{R}^3_{/\!/}$ in the following manner (Fig. 13).

In such a case, it is completely obvious that $\mathcal{P}$ is entirely characterized by the point of convergence of the segments. That point has coordinates $(\varepsilon_1, 1)_{\mathcal{R}_{/\!/}}$ in our example or, more generally, $(\varepsilon_i, \beta)_{\mathcal{R}_{/\!/}}$ in the case of $\mathcal{P}: x_i = \beta$.
We will call that point the **invariant point** of plane $\mathcal{P}$ in parallel coordinates and denote it

$$\boxed{\left|\begin{matrix}\mathcal{P}\\1,2,3\end{matrix}\right| = [\varepsilon_i : \beta : 1]}$$

Suppose now that we are in the particular case of $\mathcal{P}:\alpha_1x_1+\alpha_2x_2+\alpha_3x_3=\beta$ where $\alpha_1 = 0$.

We observe that random points on $\mathcal{P}$ plot in the following way (Fig. 14).

In our example of $(\mathcal{P}):\ 0x_1+2x_2+3x_3=7$, what happens on the first axis (x_1) is uncontrolled since the cartesian equation of $\mathcal{P}$ does not involve variable x_1. On the other hand, the shortened equation $2x_2+3x_3=7$ and our initial study of a line in $\mathbb{R}^2$ explain why we observe that all lines $\left|\begin{matrix}\mathrm{M}\\2,3\end{matrix}\right|$

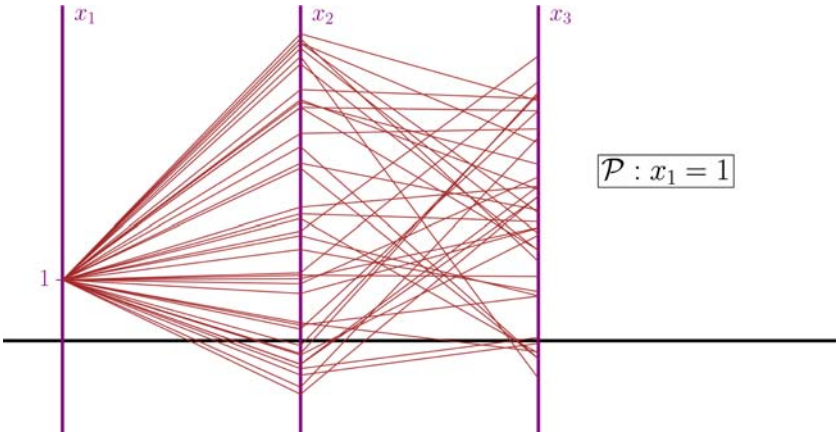


Fig. 13 Trivial plane $(\mathcal{P}) : x_1 = 1$ in $\mathbb{R}^3$

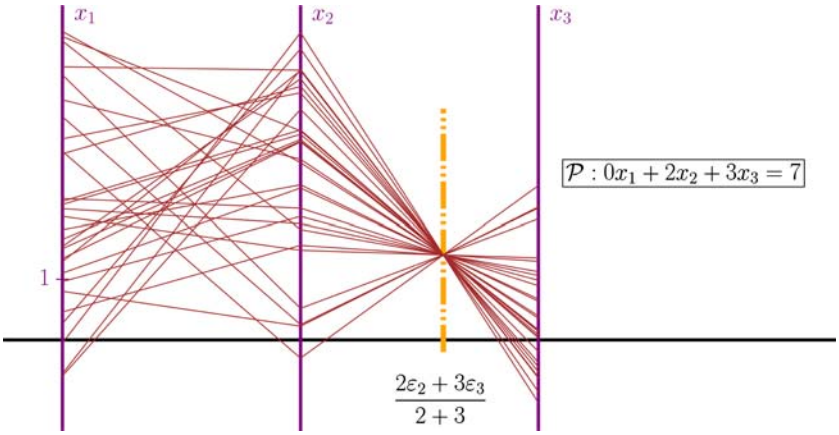


Fig. 14 $(\mathcal{P}) : 0x_1 + 2x_2 + 3x_3 = 7$

(for $M \in \mathcal{P}$) have a common point. We can even compute the (homogeneous) coordinates of that point.

$$\left|\begin{matrix}\mathcal{P}\\ 1,2,3\end{matrix}\right| = [2\varepsilon_2 + 3\varepsilon_3 : \beta : 2+3]$$

In the little more general case of $\mathcal{P} : \alpha_2 x_2 + \alpha_3 x_3 = \beta$ the invariant point would be

$$\left|\begin{matrix}\mathcal{P}\\ 1,2,3\end{matrix}\right| = [\alpha_2\varepsilon_2 + \alpha_3\varepsilon_3 : \beta : \alpha_2 + \alpha_3]$$

Now we are going to spend some time with plane $\mathcal{P} : 2x_1 + 2x_2 + 3x_3 = 13$ which offers none of the previous particularities. In a certain sense, we can consider it as a *random* affine plane of $\mathbb{R}^3$.

If we randomly select some points on it and plot these in parallel coordinates, we are bound to admit that, this time, no obvious pattern structures the plot (Fig. 15).

But if we intersect $\mathcal{P}$ with plane $x_1 = \lambda$, for different values of λ, we notice some well known patterns! (Fig. 16).

The explanation is easy to give: if x_1 is a constant, then x_2 and x_3 satisfy the equation $2x_2 + 3x_3 = (13 - 2x_1)$ and that's the equation of a line in these two variables. The **invariant point** associated with this line has (homogeneous) coordinates

$$[2\varepsilon_2 + 3\varepsilon_3 : 13 - 2x_1 : 2+3]$$

whose abscissae is independent of the constant value of x_1. As x_1 is fixed to different constant values, the **level 2 invariant point** describes a vertical line (i.e. itself parallel to the axes of the plot).

This simple observation is going to lead us to the definition of the **level 3 invariant point** associated to the plane $\mathcal{P}$.

4.2 A closer look at the case of a plane $\mathcal{P}$ in $\mathbb{R}^3$

Going one step further, we notice that the ordinate $\dfrac{\beta - \alpha_1 x_1}{\alpha_2 + \alpha_3}$ of the level 2 invariant point of $\mathcal{P} \cap \{x_1\}$ satisfies, with variable x_1, the affine equation

$$\alpha_1 x_1 + (\alpha_2 + \alpha_3)\frac{(\beta - \alpha_1 x_1)}{(\alpha_2 + \alpha_3)} = \beta$$

and in these two variables it is simply the equation of a line. So it should come equipped with an invariant point.

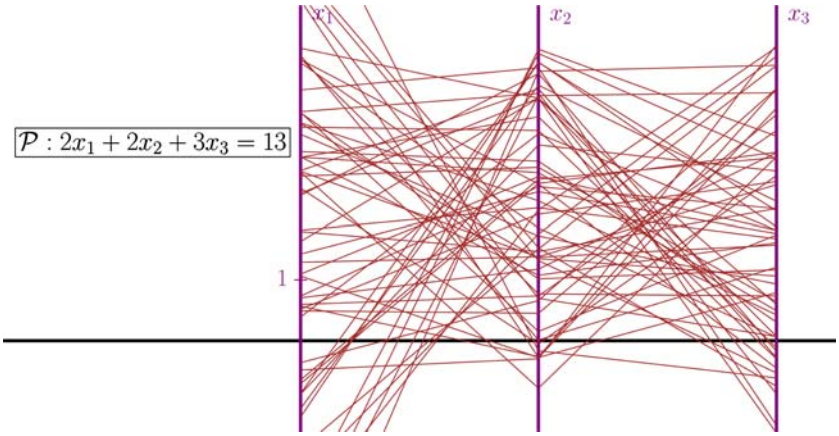


Fig. 15 Random points on $(\mathcal{P}) : \ 2x_1 + 2x_2 + 3x_3 = 13$

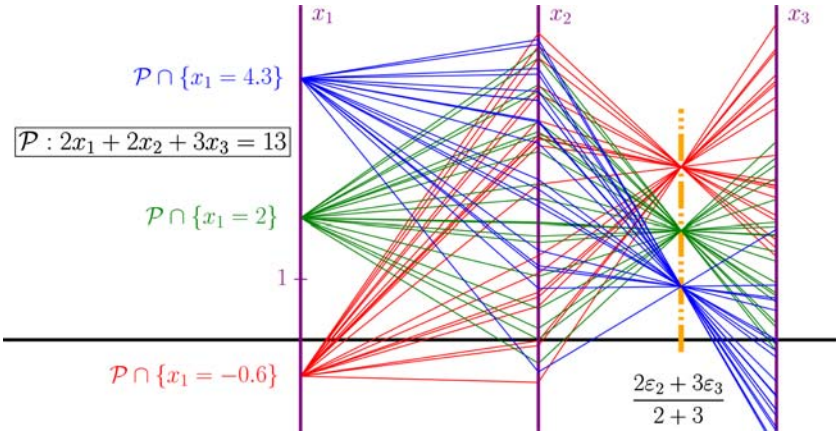


Fig. 16 Different intersections of $(\mathcal{P}) : \ 2x_1 + 2x_2 + 3x_3 = 13$ with $x_1 =$ constant

This is verified on the plot if we draw the lines going from x_1 on the first axis to the corresponding level 2 invariant point.

The point we have constructed using different level 2 invariant points will be called **the level 3 invariant point** associated with $\mathcal{P}$ (Fig. 17).

As far as it is itself a level 2 invariant point, we can use what we know about such points to compute it's homogeneous coordinates. We get, after a quick computation

$$\left|\begin{matrix}\mathcal{P}\\ 1,2,3\end{matrix}\right| = [\alpha_1\varepsilon_1 + \alpha_2\varepsilon_2 + \alpha_3\varepsilon_3 : \beta : \alpha_1 + \alpha_2 + \alpha_3]$$

Now we have to tell whether this point has interesting properties or not and if it is useful to help us decide whether a given point is in $\mathcal{P}$ or not.

Remark The previous construction of $\left|\begin{matrix}\mathcal{P}\\ 1,2,3\end{matrix}\right|$ is not possible if $\alpha_2 + \alpha_3 = 0$ because of division by zero. Later in this paper, we will give a different construction of the level 3 invariant point, helping us to avoid such accidents.

4.3 A different construction of the level 3 invariant point of a plane

We study a plane $\mathcal{P} \subset \mathbb{R}^3$ given by

$$\mathcal{P} : \quad \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = \beta$$

Let A, B, C be three noncollinear points of $\mathcal{P}$. The plane $\mathcal{P}$ is completely determined by these three points.
As we already know, plotting A, B, C in //-coordinates does not yield any clear geometric pattern of the kind we had in the case of a line in $\mathbb{R}^2$ (Fig. 18):

What actually happens is that there still is a geometric pattern or **geometric organisation** in $\mathcal{R}^3_{/\!/}$ when plotting points all belonging to a fixed plane $\mathcal{P}$, but it is not as obvious as seen before.

To show this, we will use geometric constructions of points, starting from A, B, C and not the cartesian equation.

Definition Let $\left|\begin{matrix}\mathrm{AB}\\ 1,2\end{matrix}\right|$ be the unique point of intersection of the lines $\left|\begin{matrix}\mathrm{A}\\ 1,2\end{matrix}\right|$ and $\left|\begin{matrix}\mathrm{B}\\ 1,2\end{matrix}\right|$. This point always exists in the

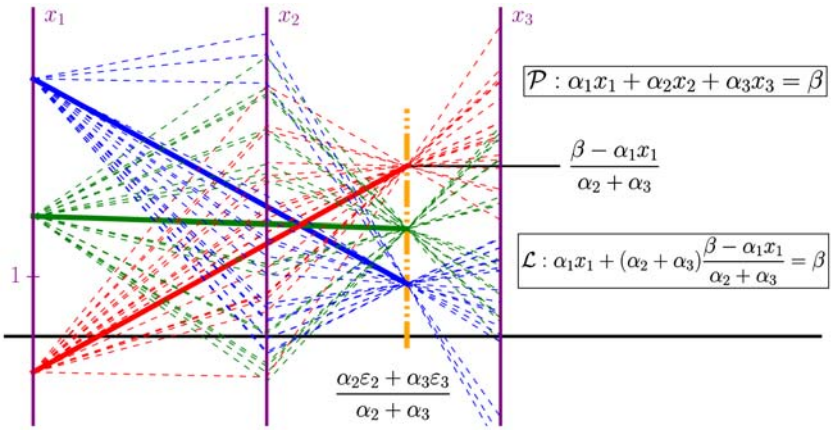


Fig. 17 How level 3 invariant point appears for $(\mathcal{P}) : \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = \beta$

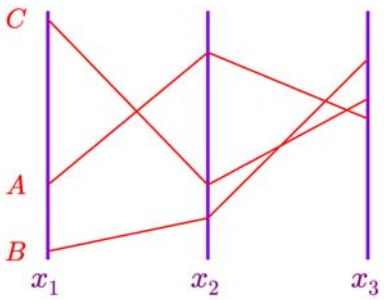


Fig. 18 Three points A, B, C in $\mathbb{R}^3$

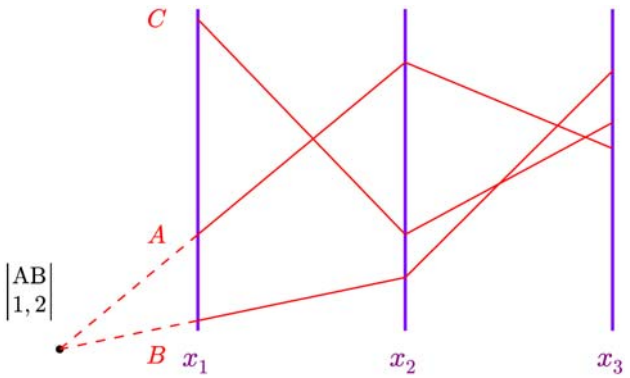


Fig. 19 Construction of the first level 2 point

projective plane holding $\mathcal{R}^3_{/\!/}$ even if the two lines are parallel (Fig. 19).

For such an intersection point, we use the notation

$$\boxed{\left|\begin{matrix}\mathrm{AB}\\1,2\end{matrix}\right| = \left|\begin{matrix}\mathrm{A}\\1,2\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{B}\\1,2\end{matrix}\right|}$$

Proposition 1 *With all preceding notations we have*

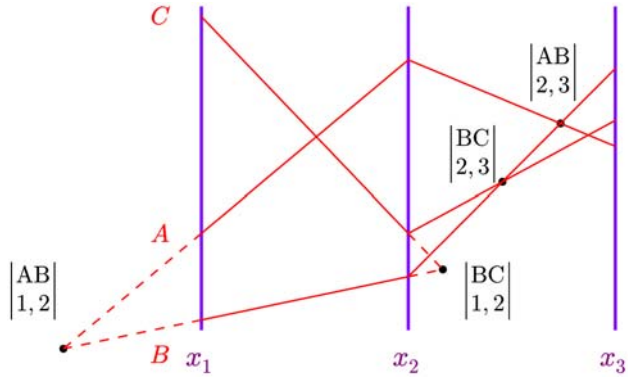


Fig. 20 Construction of four level 2 points

$$\left|\begin{matrix}\mathrm{AB}\\1,2\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [(b_2 - a_2)\varepsilon_1 - (b_1 - a_1)\varepsilon_2 : b_2 a_1 - a_2 b_1 : (b_2 - b_1) - (a_2 - a_1)]$$

Of course, in $\mathcal{R}^3_{/\!/}$, the same construction can be done with axes 2 and 3, and with the points B and C. For example, $\left|\begin{matrix}\mathrm{BC}\\2,3\end{matrix}\right| = \left|\begin{matrix}\mathrm{B}\\2,3\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{C}\\2,3\end{matrix}\right|$.

If we add these points to the original plot, we obtain (Fig. 20): For the final step of our geometric construction, we need to draw the line passing by $\left|\begin{matrix}\mathrm{AB}\\1,2\end{matrix}\right|$ and $\left|\begin{matrix}\mathrm{AB}\\2,3\end{matrix}\right|$. We will call this line $\left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right|$ (Fig. 21):

$$\boxed{\left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left(\left|\begin{matrix}\mathrm{AB}\\1,2\end{matrix}\right| \left|\begin{matrix}\mathrm{AB}\\2,3\end{matrix}\right|\right)_{\mathcal{R}_{/\!/}}}$$

Now let $\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|$ (Fig. 22) be defined by

$$\boxed{\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{BC}\\1,2,3\end{matrix}\right|}$$

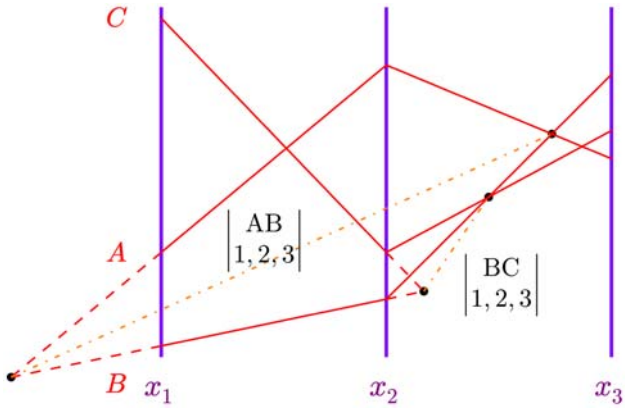


Fig. 21 Preparing for level 3 point

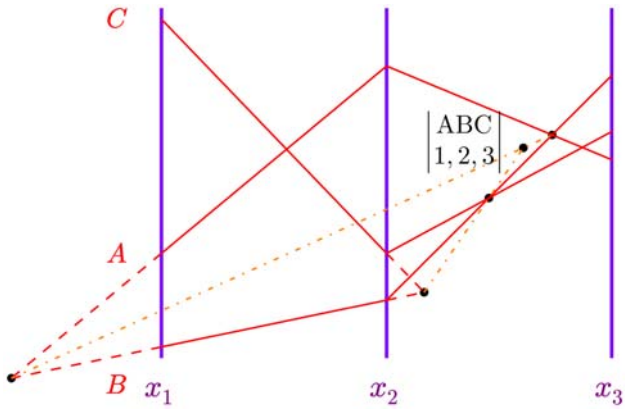


Fig. 22 Construction of a level 3 point

A simple and remarkable fact is given in the next proposition

Proposition 2 *The three lines* $\left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right|$, $\left|\begin{matrix}\mathrm{BC}\\1,2,3\end{matrix}\right|$ *and* $\left|\begin{matrix}\mathrm{AC}\\1,2,3\end{matrix}\right|$ *are convergent. For their common point of intersection, we use the following notations:*

$$\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{BC}\\1,2,3\end{matrix}\right| = \left|\begin{matrix}\mathrm{BC}\\1,2,3\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{CA}\\1,2,3\end{matrix}\right| = \left|\begin{matrix}\mathrm{CA}\\1,2,3\end{matrix}\right| \cdot \left|\begin{matrix}\mathrm{AB}\\1,2,3\end{matrix}\right|$$

This fact can be observed in Fig. 23. The Open Source software Geogebra[4] was used to create an animated construction and this image.

From the above definition, it is easy to deduce the following corollary

Corollary 1 *Let* $\sigma \in \mathfrak{S}(\{A, B, C\})$ *be any permutation on the set* $\{A, B, C\}$. *Then,*

$$\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left|\begin{matrix}\sigma(\mathrm{A})\sigma(\mathrm{B})\sigma(\mathrm{C})\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}}$$

There are many ways to prove the first proposition. For those not familiar with projective geometry and Desargues' theorem, a direct analytical computation is possible. But if no trick is used to simplify the computation, it might seem hard to do it by hand. In such a case, the open source mathematical software Sage[5] can easily be used to compute the coordinates of the level three point.

But in the end, it all boils down to a point with quite simple coordinates: the ones we obtained in our previous construction of the level 3 invariant point of $\mathcal{P}$! And that's a fundamental coincidence.

Proposition 3 (Fundamental result)

$$\boxed{\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [\alpha_1\varepsilon_2 + \alpha_2\varepsilon_2 + \alpha_3\varepsilon_3 : \beta : \alpha_1 + \alpha_2 + \alpha_3]}$$

in the case where (ABC) *is a well defined affine plane with cartesian equation*

$$\alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = \beta$$

Remark Again, it is visible from the above expression that $\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}}$ is invariant under permutation of the letters A, B, C.

Remark The coordinates of $\left|\begin{matrix}\mathrm{ABC}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}}$ only depend on the coefficients of the cartesian equation of $\mathcal{P}$ and of the abscissas $\varepsilon_1, \varepsilon_2, \varepsilon_3$ used when building $\mathcal{R}^3_{/\!/}$.

This motivates the following definition.

Definition Let $\mathcal{P} : \alpha_1 x_1 + \alpha_2 x_2 + \alpha_3 x_3 = \beta$ be a plane of $\mathbb{R}^3$.
We define **the level 3 point invariant** of $\mathcal{P}$ in $\mathcal{R}^3_{/\!/}$ as the point

$$\left|\begin{matrix}\mathcal{P}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}} = [\alpha_1\varepsilon_2 + \alpha_2\varepsilon_2 + \alpha_3\varepsilon_3 : \beta : \alpha_1 + \alpha_2 + \alpha_3]$$

Remark It is very important to understand that $\mathcal{P}$ completely determines $\left|\begin{matrix}\mathcal{P}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}}$ but the reciprocal is false.

Obviously, the knowledge of $\left|\begin{matrix}\mathcal{P}\\1,2,3\end{matrix}\right|_{\mathcal{R}_{/\!/}}$ is not enough to recover the original cartesian equation of $\mathcal{P}$. At least, you need the data of a point A in $\mathcal{P}$.

It can be shown that it is then enough to recover $\mathcal{P}$.

[4] http://www.geogebra.org/.

[5] http://www.sagemath.org.

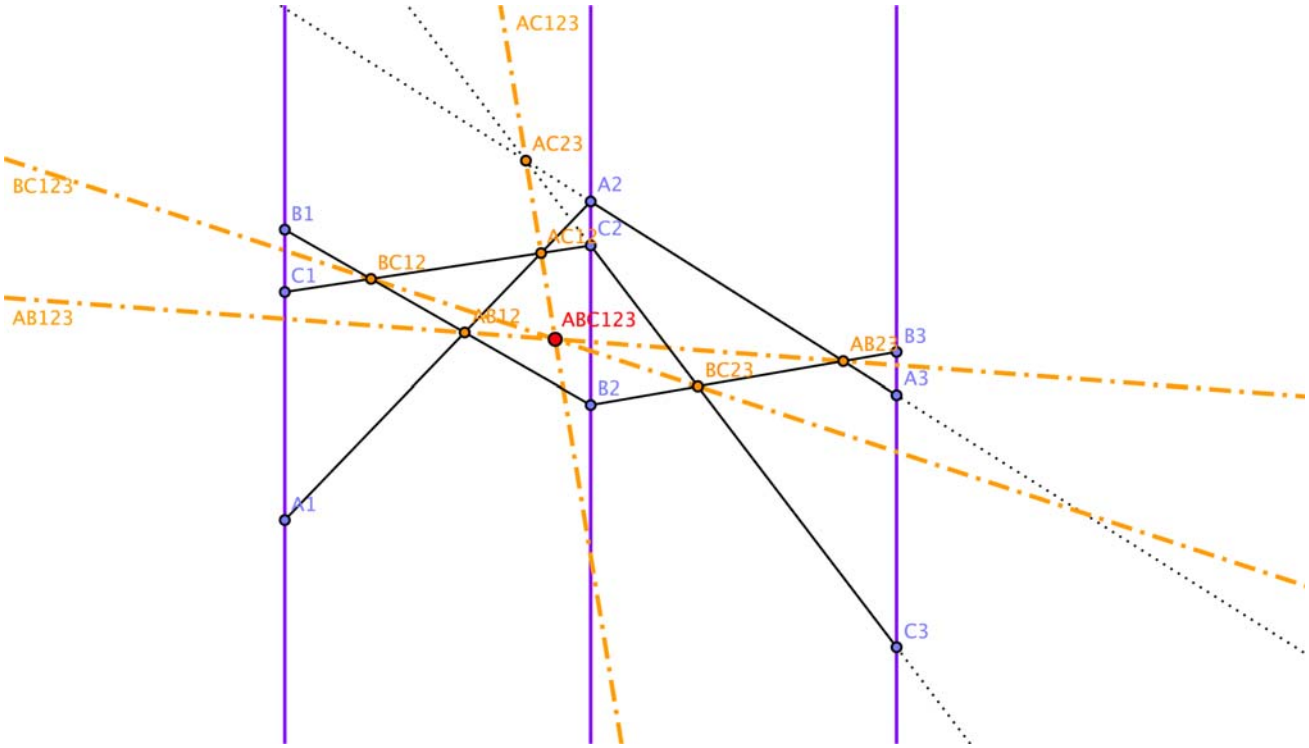


Fig. 23 Constructions of level two and level three points for (ABC)

5 Geometrical proof of Proposition 2

We will begin with a short review of projective geometry.

5.1 A quick introduction to projective geometry

Projective geometry only considers **points** and **lines** and does not use notions as distance or angle. It is based (classically) on 4 axioms:

- By two points, there always is a line passing, which is unique if the two points are different
- Two lines always meet
- It is possible to find 4 points such that any three of them are never aligned
- Pappus' theorem

Pappus' theorem has many different expressions but an easy one can be read from the following figure
Let (ABC) and $(A'B'C')$ be two triangles such that $C \in (A'B')$ and $C' \in (AB)$.
Let $i = (AC) \cap (A'C')$, $j = (BC) \cap (B'C')$ and $k = (AB') \cap (A'B)$.
Then i, j, k are aligned.
One of the first consequence of Pappus' theorem is the famous Desargues' theorem.
Consider the following figure (Figs. 24, 25)

Desargues theorem states that if (ABC) and $(A'B'C')$ are two triangles such that (AA'), (BB') and (CC') meet in a common point O, then the points of intersection of sides of both triangles having same names are aligned.
More precisely, if $i = (AB) \cap (A'B')$, $j = (AC) \cap (A'C')$, $k = (BC) \cap (B'C')$, then i, j, k are on a same line.

By a very important duality principle in Projective Geometry, Desargues' theorem admits a reciprocal. This reciprocal states that, with the preceding notations, if i, j, k are aligned then (AA'), (BB') and (CC') meet in a common point (Fig. 26).

Using this result, it is possible to prove the first proposition concerning the level 3 invariant point.

5.2 A geometrical proof of Proposition 2

Recall that our geometric construction of $\begin{vmatrix} \text{ABC} \\ 1, 2, 3 \end{vmatrix}$ is based on the following process.
We first compute the six points obtained by intersecting lines in-between the same axes (Fig. 27).
Then, we pretend that the three lines $\begin{vmatrix} \text{AB} \\ 1, 2, 3 \end{vmatrix}$, $\begin{vmatrix} \text{BC} \\ 1, 2, 3 \end{vmatrix}$ and $\begin{vmatrix} \text{AC} \\ 1, 2, 3 \end{vmatrix}$ are convergent (Fig. 28).
To prove this, we will simplify the drawing and only keep the central axis (Fig. 29)

Fig. 24 Pappus' theorem

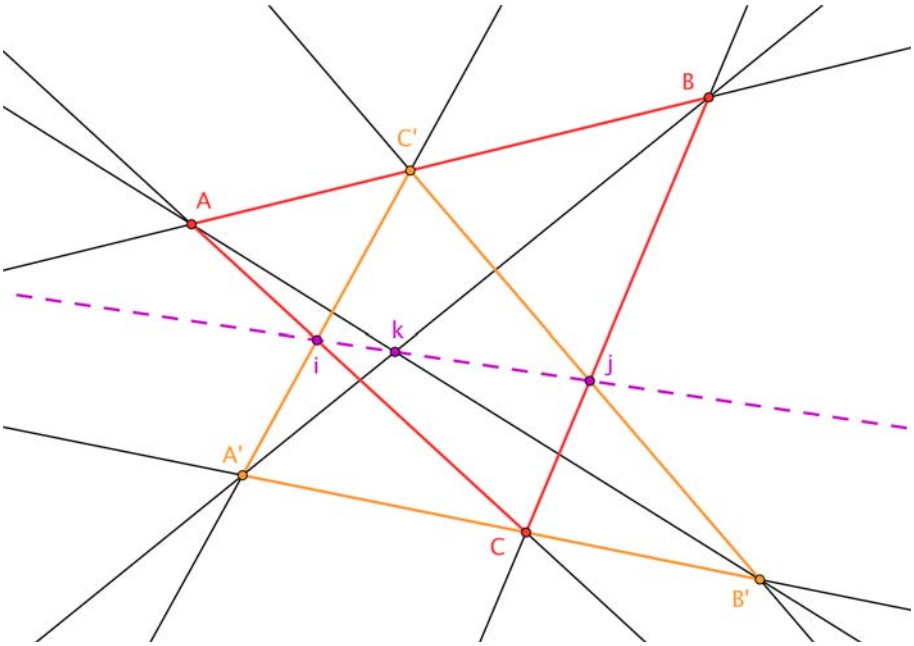
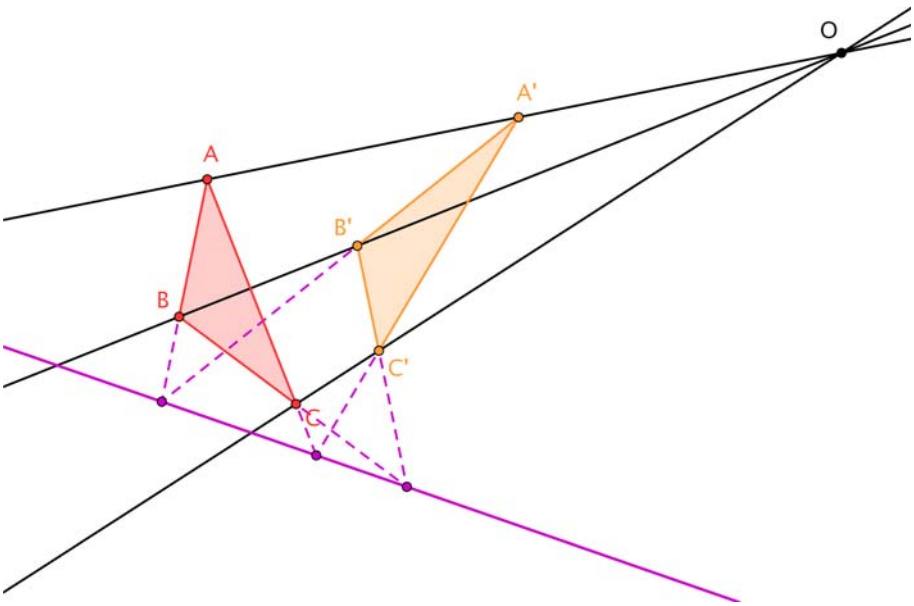


Fig. 25 Desargues' theorem



We notice two triangles satisfying the hypothesis of the reciprocal of Desargues' theorem and therefore can conclude that our three lines meet at a same point (Fig. 30).

6 Hyperplanes in $\mathbb{R}^N$

It is possible to generalize the preceding constructions and define points associated to general hyperplanes H of $\mathbb{R}^N$.

Definition 6.0.1 Let $H : \alpha_1 x_1 + \alpha_2 x_2 + \cdots + \alpha_N x_N = \beta$ an affine hyperplane of $\mathbb{R}^N$.

Let $\mathcal{R}^N_{/\!/}(\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_N)$ be a fixed $/\!/$-coordinate system. The $/\!/$-coordinate **level** N **point associated with** H **in** $\mathcal{R}^N_{/\!/}$ is:

$$\left| \begin{matrix} \mathrm{H} \\ 1, 2, \ldots, N \end{matrix} \right|_{\mathcal{R}^N_{/\!/}} = [\alpha_1\varepsilon_1 + \cdots + \alpha_N\varepsilon_N : \beta : \alpha_1 + \cdots + \alpha_N]$$

The next result shows that level N points can be build recursively by a simple geometric process from four level $N-1$ points.

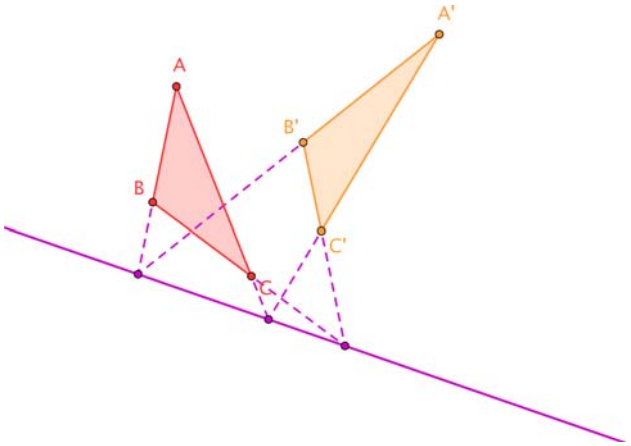


Fig. 26 Reciprocal of Desargues' theorem

Proposition 4 *For any family of N points* $(A_1, A_2, \ldots, A_N)$ *defining hyperplane H, one has:*

$$\left|\begin{matrix} \mathrm{H} \\ 1, \ldots, N \end{matrix}\right|_{\mathcal{R}_{/\!/}} = \left|\begin{matrix} \mathrm{A_1 A_2 \ldots A_N} \\ 1, 2, \ldots, N \end{matrix}\right|_{\mathcal{R}_{/\!/}}$$

where the last point is defined recursively by:

$$\left|\begin{matrix} \mathrm{A_1 A_2 \ldots A_N} \\ 1, 2, \ldots, N \end{matrix}\right| = \left(\left|\begin{matrix} \mathrm{A_1 \ldots A_{N-1}} \\ 1, \ldots, N-1 \end{matrix}\right| \left|\begin{matrix} \mathrm{A_1 \ldots A_{N-1}} \\ 2, \ldots, N \end{matrix}\right|\right) \cdot \left(\left|\begin{matrix} \mathrm{A_2 \ldots A_N} \\ 1, \ldots, N-1 \end{matrix}\right| \left|\begin{matrix} \mathrm{A_2 \ldots A_N} \\ 2, \ldots, N \end{matrix}\right|\right)$$

The whole process starts with the N^2 *points* $\left|\begin{matrix} \mathrm{A_i} \\ j \end{matrix}\right|$.

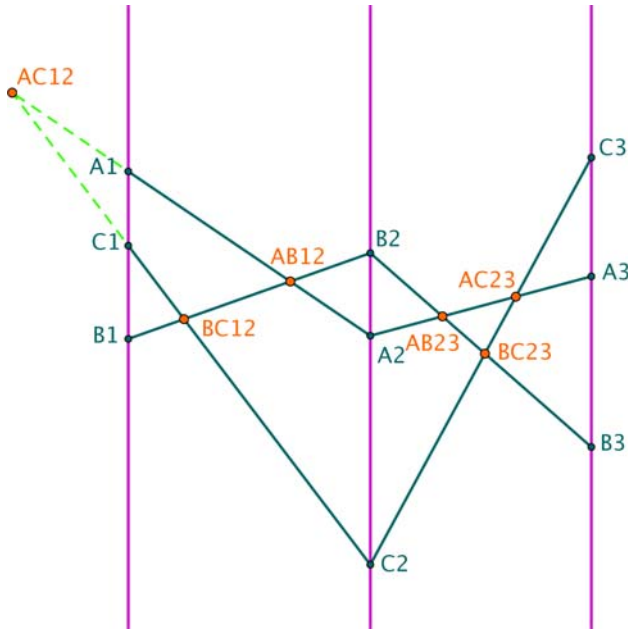


Fig. 27 Construction of the level 3 invariant point

Remark It is very important to note that it is not an easy computation to prove that both definitions of the level N invariant point coincide.

To be more precise, the fact that the recursive process produces, in the end, a point with coordinates that can be easily expressed in terms of the coefficients of some cartesian equation of the hyperplane looks like a beautiful algebraic results to both autors.

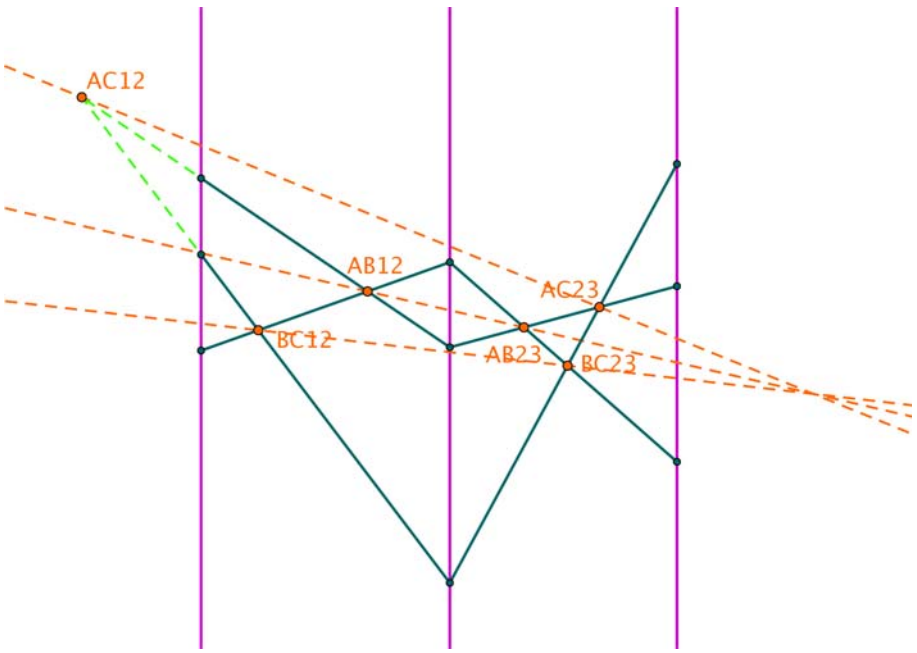


Fig. 28 Construction of the level 3 invariant point

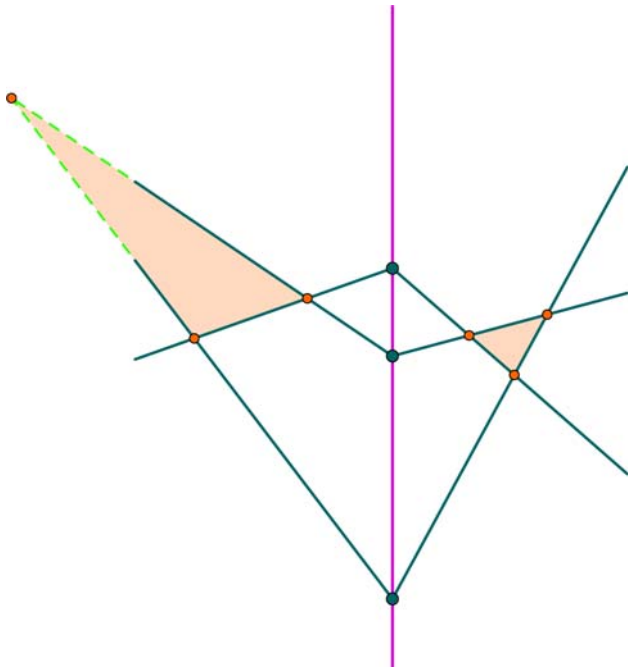


Fig. 29 Simplifying the figure

7 Detection of more general geometric structures with //-coords

As we want to keep this mathematical introduction quite elementary, our journey in the theoretical //-coords universe will stop shortly. We hope that the very basic aspects of //-coords are now familiar to you and that you understand better how classical affine geometric objects can be detected.

Of course, we know that it is not enough to detect linear structures and that in real life, the mathematical relationship between variables can be more complex. It is therefore essential to know wether //-coordinates are able or not to deal with such datasets.

Furthermore, it is one thing to examine a pure mathematical object defined by a set of algebraic equations in $\mathbb{R}^N$ (as in real algebraic geometry), and it is another to manage experimental data that are supposed to obey a certain class of mathematical laws. It is even a third story to investigate computer security logs given that it is very doubtful that any analytical relationship exists between the variables.

In a forthcoming article, we hope to report about a **classifier** based on //-coords and pioneer work of Inselberg [6–9] and some new ideas. This classifier, hopefully, will show how //-coords can go much beyond our actual exposition.

8 Automating //-coords creation with Picviz

Now that the theory behind //-coords is known, and since Picviz goal is to automate the creation of //-coords images out of logs, this section introduces its architecture and how it can be used to discover security issues. Picviz is not limited to computer security, however since it is a good goal to demonstrate how powerful can be such a tool, this is what the paper sticks to.

Because digging visually for security issues is aimed to be very practical, Picviz presentation starts with an authentication log investigation. After this short presentation, the architecture is detailed. Once the reader knows features provided by the software, two examples are given.

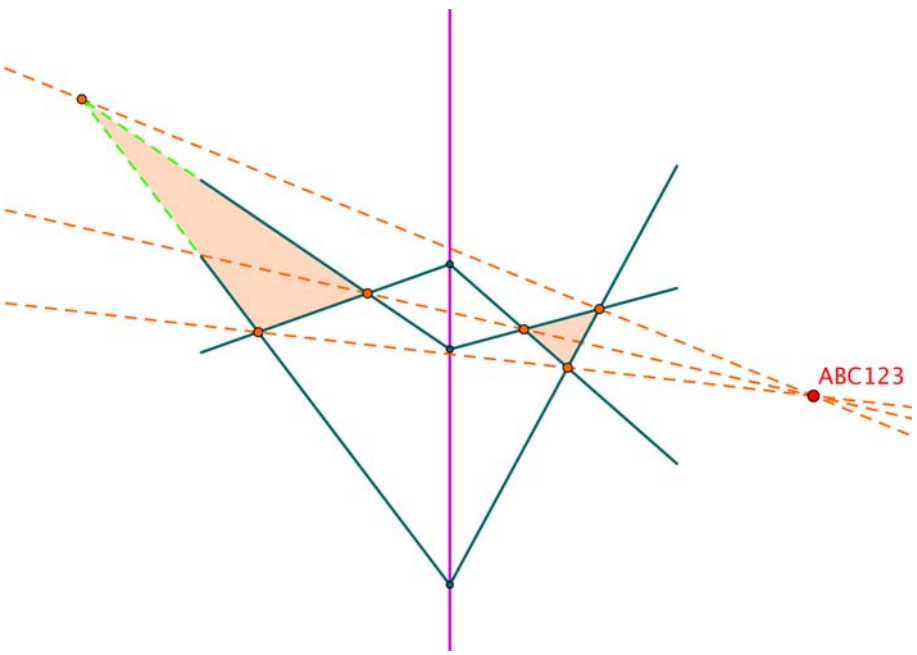


Fig. 30 How we apply the reciprocal of Desargues' theorem

8.1 Picviz

Picviz[6] is a software transforming acquired data into a parallel coordinates plot image to visualize data and discover interesting results quickly. Picviz is composed in three parts, each facilitating the **creation**, **tuning** and **interaction** of //-coords graphs (Fig. 31):

- **Data acquisition**: log parsers and enricher
- **Console program**: transforms PGDL into a svg or png image. Unlike the graphical frontend, it does not require graphical canvas to display the lines, it is fast and able to generate millions of lines.
- **Graphical frontend**: transforms PGDL into a graphical canvas for user interaction.

It was written because of a lack of visualization tools able to work with a large set of data. Graphviz is very close to how Picviz works, except that is has limitations regarding the number of dimensions that can be handled by a directed graph, such as when dealing with time.

8.2 Data acquisition

The data acquisition process aims to transform captured logs into the Picviz Graph Description Language (PGDL) file before Picviz treatment. In this paper, log is used interchangeably with data to express something that is captured from one or several machines. That being one in:

- **Syslog**: System and application log files. Containing at least four variables: time, machine, application and the logged event.
- **Network**: Sniffed data.
- **Database**: Structured information storage.
- **Specific**: Log file for applications not using standard log functions.
- **Other**: Any other way to record events.

CSV being a common format to read and write such data, Picviz can take it as input and will translate it into PGDL.

8.3 Console program

The Command Line Interface (CLI) allows Picviz to easily generate //-coords graphs, including frequency analysis, filtering and powerful filters. As the CLI allows to deal with millions of events, this may be the first step prior using the graphical frontend. The CLI is the **pcv** program that is being used in this paper to show how graphs were generated.

[6] This paper describes Picviz version 0.6.x.

8.4 Graphical frontend

While the Graphical frontend does not allow to deal with as much objects as the CLI does, it still gives valuable insight for the analyst. Live interaction is something the frontend can provide:

- moving the mouse over the lines will actually show the lines values being displayed,
- changing the axis order on the fly by selecting the appropriate value on the axis description box,
- zooming to allow line-to-line relationship finding (Fig. 32),
- slider to move in the event time scale: moving the cursor to the left-most will show the first event,
- brushing to find the relationships in multiple dimensions of a given event (Fig. 33).

8.5 Understanding 10000 lines of log

Visualization is an answer to analyze a massive amount of lines of logs. //-coords helps to organize them and see correlations and issues by looking at the generated graph [3].

Each axis strictly defines a variable: logs, even those that are unorganized, are always composed by a set of variables. At least they are: **time** when the event occurred, **machine** where the log comes from, **service** providing the log information, **facility** to get the type of program that logged, and the **log** itself.

The **log** variable is a string that varies widely based on the application writing it and what it is trying to convey. This variability of the string is what makes the logs disorganized. From this string, other variables can be extracted: username, IP address, failure or success, etc.

Log sample: PAM session

```
Aug 11 13:05:46 quinificated su[789]:
 pam_unix(su:session): session opened
 for user root by toady(uid=0)
```

Looking at the PAM session log, we know how the user authenticates with the common **pam_unix** module. We know that the command **su** was used by the user **toady** to authenticate as **root** on the machine **quinificated** on **August 11th at 1:05 p.m.**. This is useful information to care about when dealing with computer security. In this graph we clearly identify:

- If a user sometime fails to give the correct password
- If a user logged in using a noncommon pam module or service
- Time when users log in

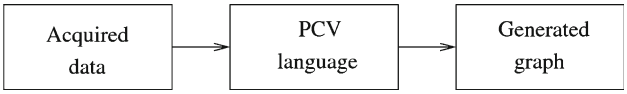


Fig. 31 Picviz simplified architecture

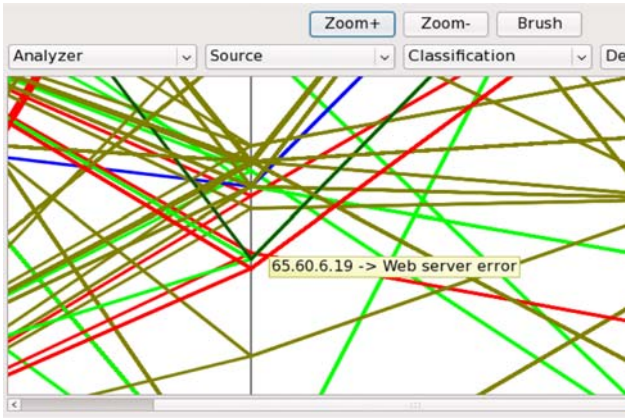


Fig. 32 Graphical frontend zoom feature

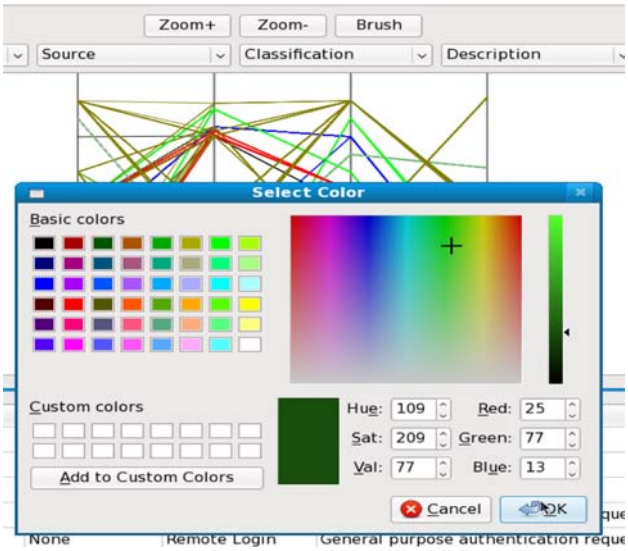


Fig. 33 Graphical frontend brushing feature

Figure 34 shows the representation of the **auth** syslog facility:

Analysis

What one can easily see in Fig. 34 is how many users logged in as root on the machine: red lines means root destination user. Also, the leftmost axis (**time**) is interesting: it has a blank area and using the frontend we discover that no one opened a session between 2:29 a.m. and 5:50 a.m.:

The **second axis** is the **machine** where the logs originated. Since this example is a single machine analysis, lines converge to a single point.

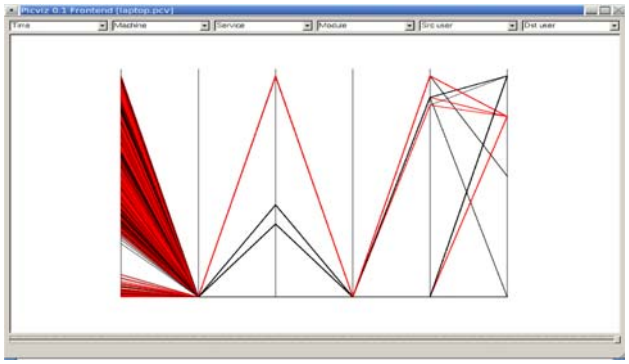


Fig. 34 Picviz frontend showing pam sessions opening

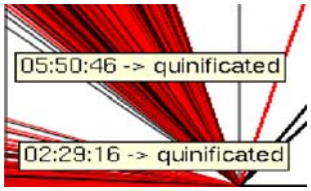


Fig. 35 Zoom on time axis

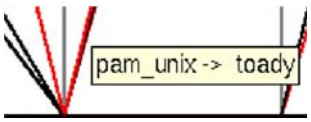


Fig. 36 PAM module convergence

This **third axis** is the **service** or application that wrote the log. We can quickly see four services (one red, three blacks, the line at the bottom is also a connection between two axes): moving the mouse above the red line at the service on top of Fig. 35 shows that only the 'su' service is used to log a user as root. Hopefully no one logs in using **gdm**, **kdm** or **login** as root.

The **fourth axis** is the **pam module** that was used to perform the login authentication: again, as only local authentication was done using the **pam_unix** module, lines are converging. If we would have had a remote authentication, or other modules opening the session, we would see them on this axis (Fig. 36).

The **fifth and sixth** axis are the user source and destination of the logs. We have as much source logins as destination logins. On this machines, logins are both **su** and **ssh**.

As experts might know, //-coords are already used in computer security [1] but face a problem of not being easy to automate or with various data formats. This paper focuses on how relevant security information can be extracted from logs, whatever format they have, how anyone can discover attacks

or software malfunctions and how the analyst can then filter and dig into data to discover high level issues. The next part covers how Picviz was designed, its internals. After we will see how malicious attacks can be extracted, and how it can help you to write correlation rules.

9 Picviz architecture in detail

9.1 Picviz graph description language

It has been designed to be easy to generate and as close as possible to the Graphviz [2] language (mostly for properties names). It is a description language for //-coords which allows to specify all kinds of properties for a given line (**data**), set each axis variable **axes** and give instructions to the engine in the **engine** section. Also, a graph title can be set in the **header** section.

Below is an example of a PGDL source which represents a single line:

```
header {
 title = "foobar";
}
axes {
 integer axis1 [label="First"];
 string axis2 [label="Second"];
}
data {
 x1="123",x2="foobar" [color="red"];
}
```

Despite CSV being a standard for log normalization in order to create a graph, and even though Piviz can convert CSV into PGDL, CSV is not recommended. It is impossible to set properties on an individual line when a specific item is encountered. This would require an external configuration file to be parsed at every new added line. This would greatly decrease performances. Also, //-coords have the weakness of hiding interesting values according to the axis order. Changing the axis order in PGDL is as simple as moving a line in the **axes** section.

9.1.1 The axes section

It defines possible types you can set to each axis, as well as setting axis properties. Labels can be set to axes with the **label** property. Axes types must be one of them:

Type	Range	Description
timeline	"00:00" - "23:59"	24 h time
years	"1970-01-01 00:00" - "2023-12-31 23:59"	Several years
integer	0 - 65535	Integer number
string	"" - "No specific limit"	A string
short	0 - 32767	Short number
ipv4	0.0.0.0 - 255.255.255.255	IP address
gold	0 - 1433	Small value
char	0 - 255	Tiny value
enum	anything	Enumeration
ln	0 - 65535	Log(n) function
port	0 - 65535	Special way to display port numbers

It is indeed possible to specify the maximum value an axis can get as a numeric value instead of the axis type. This would allow value *1234* to be the maximum of *axis1*:

```
axes {
   1234 axis1;
   ...
}
```

Other available properties are:

- **print**: when set to false, removes values printing on lines. It is usually used when an axis had too big values which are overlapping the next axis.
- **relative**: when set to true place values on the axis relatively to each other. Which decreases performances but can improve the axis reading.

9.1.2 The data section

Data are written line by line, each value coma separated. Four data entries with their relatives axes can be written like this:

```
data {
 t="11:30",src="192.168.0.42", dport="80"
  [color="red"];
 t="11:33",src="10.0.0.1", dport="445";
 t="11:35",src="127.0.0.1", dport="22";
 t="23:12",src="213.186.33.19", dport="31337";
}
```

The *key*=*value* pair allows to identify which axis has which value. Since axis variable type was defined in the previous **axis** section, the order does not matter. For the previous data, if one want to respect the order, the **axis** section would be:

```
axes {
   timeline t;
   ipv4     src;
   port     dport;
}
```

Changing the axis order and repeating the axes is explained in the **Grand tour** section.

Every line can receive two properties: **color** and **penwidth**, which allow to set the line color and width, respectively.

Data lines are generated by scripts from various sources, ranging from logs to network data or anything a script can capture and transform into PGDL language data section. This paper focuses on logs, and Perl was chosen for its convenience with Perl compatible regular expressions (PCRE) built-in with the language. The next part explains how such a script can be written to generated the PGDL.

9.2 Generating the language

Picviz delivers a set of tools to automate the PGDL generation from various sources, such as **apache** logs, **iptables** logs, **tcpdump**, **syslog**, SSH connections, …
Perl being suited language for this kind of task, it was chosen as the default generator language. Of course, nothing prevent other people to write generators for their favorite language.

The PGDL is generated with the Perl **print** function, along with Perl pattern matching capabilities to write the data section. The syslog to PCV takes 25 lines of code, including lines colorization where the word 'segfault' shows up in the log file. Then, to use the generator, type:

```
perl syslog2pcv /var/log/syslog > syslog.pgdl
```

To help finding evilness, a Picviz::Dshield class was written. Calling it will check if the port or IP address match with dshield.org database:

```
use Picviz::Dshield;

dshield = Picviz::Dshield->new;

ret = dshield->ip_check("10.0.0.1")
```

It can be used to set the line color, to help seeing an event correlated with dshield information database.

9.3 Understand the graph

9.3.1 Graphical frontend

Aside from having a good looking graph, it is good to dig into it, and see what was generated. An interactive frontend was written for this purpose. It is even a good example on how Picviz library can be **embeded** in a Python application. The application **picviz-gui** was written in Python and Trolltech QT4 library.

The frontend provides a skillful interaction to find relationship among variables, allows to apply filters, drag the mouse over the lines to see the information displayed and to see the time progression of plotted events. Real-time capabilities are also possible, since the frontend listen to a socket waiting for lines to be written.

The frontend has limitations: on a regular machine, more than 10000 events makes the interface sluggish. As Picviz was designed to deal with million of events, a console program was written.

9.3.2 Command line interface

The **pcv** program is the CLI binary that takes PGDL as input, uses the picviz library output plugins and generate the graph using called plugin. To generate a SVG, the program can be called like this:

```
pcv -Tsvg syslog.pgdl > syslog.svg
```

As SVG is a XML and vectorial format, it will perform well when a few thousand of line are drawn: the operator will be able to do actions on items, select them, grep for a certain value, move the lines, etc.

However, with a big set of data, SVG frontend will fail doing the rendering.

This is why a PNG capable plugin was written. Using the cairo[7] library. The plugin is named **'pngcairo'** and can be used like this:

```
pcv -Tpngcairo syslog.pgdl > syslog.png
```

Usually is it better to use the PNG plugin, filter data and if needed, use then choose between the SVG plugin to use all the features a vectorial image can provide or the Picviz frontend, that is designed to deal with //-coords issues. Section **2.3.4** explains how filters can be used with Picviz.

9.3.3 Grand tour

Because choosing the right order for the right axis is one of //-coords disadvantages, Picviz provides via the **pngcairo** plugin a **Grand tour** capability. The **Grand tour** generates as much images as pairs permutation of axes possible, the idea is to show every possible relation among every available axes. Plugin arguments are provided with the **-A** command. So to generate a grand tour on graphs, **pcv** should be called like this (Fig. 37):

```
pcv -Tpngcairo syslog.pcv -Agrandtour
...
File Time-Machine.png written
File Time-Service.png written
...
File Log-Machine.png written
File Log-Service.png written
```

[7] http://www.cairographics.org.

Fig. 37 Syslog grand tour

An other way to change the axes order is simply changing their order in the **axes** section.

```
axes {
 integer axis1;
 string  axis2;
 ipv4    axis3;
}
```

Shows in the order: axis1, axis2 and axis3, while:

```
axes {
 ipv4    axis3;
 integer axis1;
 string  axis2;
 ipv4    axis3;
}
```

will show the axes in the order axis3, axis1, axis2 and then axis3 again. Thus changing the axes–pair relationship. This section also allows commenting to hide a given axis.

It is recommended to have the data separated from the image and axes properties to allow easy changing, this can be done using the **@include** keyword, that will include data from a third-party file. In the case of numerous data, this is very convenient to find relationships and allow easy changing of axis type and order:

```
axes {
 ipv4    axis3;
 integer axis1;
 string  axis2;
}
data {
@include "mydata.pgdl"
}
```

9.3.4 Real-time

Picviz can be set up to perform real-time line drawing by listening to a socket. Then programs can send their lines to this socket.

When Picviz listen to a socket, it should have a template associated with it. This is a graph written in a template derived from PGDL named PGDT for Picviz Graph Description Template. The difference with a PGDL file is that a template may not have any data, so that the program will know which variables are associated with the axes. Of course PGDT is more convenient to express a file that is created for real-time, but it can interchangeably be used with PGDL.

To start PCV in real-time mode, one can start on one side:

```
pcv -s local.sock -t samples/test1.pgdl
-Tpngcairo -o /tmp/graph.png
```

And on the other side, the client will simply need to write onto this socket:

```
echo "t=\"12:00\", i=\"100\",
 s=\"I write some stuff\";"
```

The OSSEC HIDS[8] can output its alerts to Picviz, using the template **ossec.pgdt** provided with sources.

9.3.5 Filtering

To select lines one want to be displayed, Picviz provides filters. They can be used on the real value to match a given regular expression, line frequency, line color or position as mapped on the axis. It is a multi-criterion filter. It is set with the CLI or Frontend parameters.

With the CLI, they can be called like this:

```
pcv -Tpngcairo syslog.pcv 'your filter here'
```

With the frontend, filter can be called like this:

```
picviz-gui syslog.pcv 'your filter here'
```

Filter syntax is:

```
display type relation value on axis
 number && ...
```

Where:

- **display**: show or hide, select if we hide or display the selected value
- **type**: value, plot, color or freq, choose what is filtered
- **relation**: <, >, <=, >=, ! =, =, relation with selected value

[8] http://www.ossec.net.

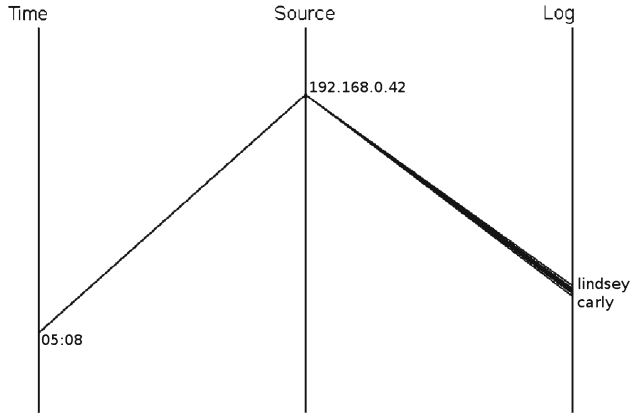


Fig. 38 SSH scan

- **value**: selected value to compare data with
- **on axis**: text to express the axis selection
- **number**: axis number to filter values on

For example, to display all lines plotted under a hundred on the second axis, one can replace **your filter here** by **plot < 100 on axis 2**. Specific data can also be removed, such as:

```
pcv -Tpngcairo syslog.pcv 'value = "sudo"
 on axis 2'
```

A percentage can be applied to avoid knowing the value that can be filtered: **'plot >(42% on axis 3 and plot < 20% on axis 1'**.

9.3.6 String positioning

One of the basic string algorithm displaying is to simply add the ascii code to create a number. Among pros of using this very naive algorithm, is to be able to display scans (strings very close to each other coming from one source) very easily. As for the cons there is the collision risk, but in practice this low risk of having such events. As Picviz is very flexible, it still offer other string alignment algorithms, using Levenstein [10] or Hamming distance [4] from a reference string. This still makes collision possible, but differently.

The basic algorithm highlights scans evidences, and then one can quickly spot an issue. This way, without having any knowledge of how the log must be read, little changes will appear close enough to each other to grab the reader attention.

The following lines are logs taken from ssh authentication, and appear like this:

```
time="05:08", source="192.168.0.42",
 log="Failed \ keyboard-interactive/pam
 for invalid user lindsey";
time="05:08", source="192.168.0.42",
 log="Failed \ keyboard-interactive/pam
```

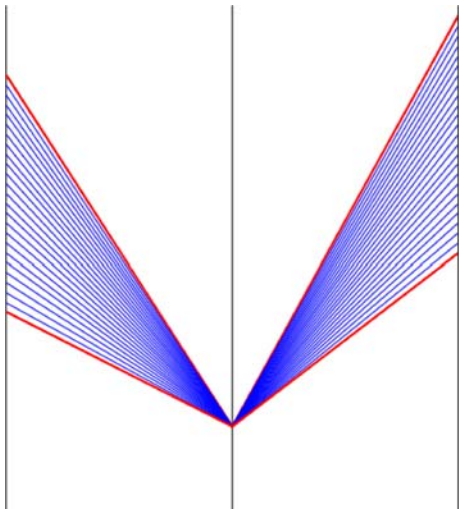


Fig. 39 Same shared value

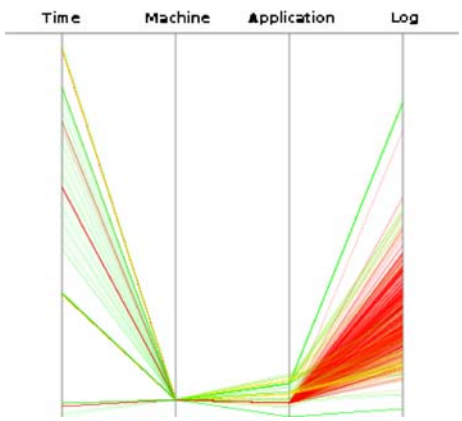


Fig. 40 Syslog heatline with virus mode

```
 for invalid user ashlyn";
time="05:08", source="192.168.0.42",
 log="Failed \ keyboard-interactive/pam
 for invalid user carly";
```

Figure 38 shows a generated graph from twenty lines of a ssh scan.

On the third axis, one can clearly see the lines sweep, showing the scan.

9.3.7 Correlations

With //-coords, several correlations are possible, as shown in Fig. 39, where it is known for sure all events share a common variable.

One other way to correlate is applying a line colorization for their frequency of apparition between two axes and colorize the whole line, according to the highest value. Picviz can generate graphs in this mode with the **heatline** rendering plugin and its **virus** mode (Fig. 40).

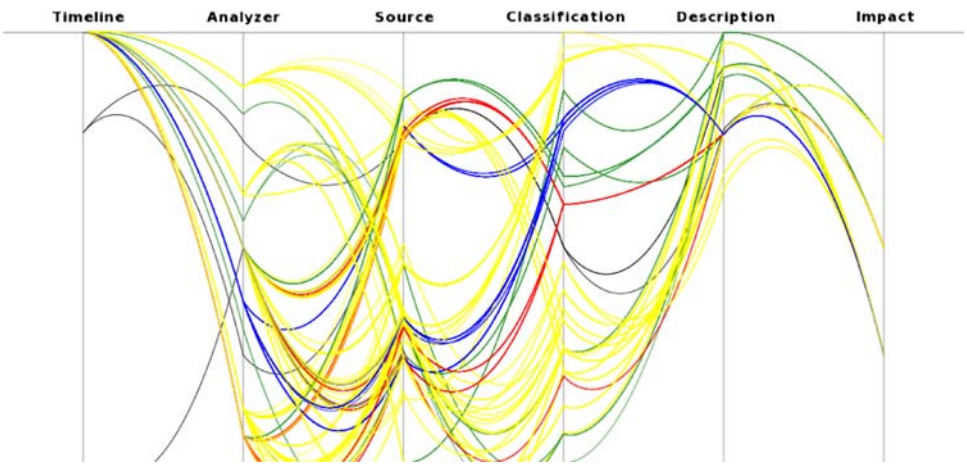


Fig. 41 Picviz curves

As of today, only the svg and pngcairo handle this feature. Picviz CLI should be called like this:

```
pcv -Tpngcairo syslog.pgdl -Rheatline
 -Avirus > syslog.png
```

9.3.8 Curves

Curves is the idea of drawing arc circles instead of lines to maybe uncover clusters more easily. While this technique will go against all the mathematical theory described in this paper, some people consider this. Nothing useful has been found using this technique. It is shown to get an idea of what one can do with //-coords. See Fig. 41.

9.3.9 Section summary

Picviz has been designed to be very flexible and let anyone capable to generate the language, filter data and visualize them. This can be done statically on a plain generated file, and with the graphical interface, this is even possible in real-time. Of course the knowledge of logs lines is better to set more axis and have more information to understand the graph. However, the naive approach is sometime enough to see scanning activities.
Now that the Picviz architecture and features have been explained, we will show how we can efficiently use it to dig into logs and extract relevant information.

The next section covers how we can efficiently **see** attacks from many lines of log and finally write a correlation script in perl.

10 Understanding the unknown

This section explains how one can handle in a very short amount of time the analysis of failures a system can have based on their logs. The log issues coverage will not be exhaustive but using //-coords can easily spot some.

During the latest Usenix Workshop on the Analysis of System Logs 2008, Cray systems gave logs to analysts to see what they could extract from them.

Cray logs are available for download in the Usenix website. [9] This analysis is done on the file **0809181018.tar.gz**.

10.1 Files overview

Once uncompressed, there are sixty-seven files and four main directories, which are:

- eventlogs
- log
- SEDC_FILES
- xt

The **log** directory does not contain enough information: a single 18 lines file. As Picviz is good to manage big files, the **eventlogs** directory was chosen.

The **eventlogs** directory has two files: **eventlogs** and **filtered-eventlog**. As the last file is smaller and from its name seems to be filtered, it is best to run Picviz on **eventlogs**.

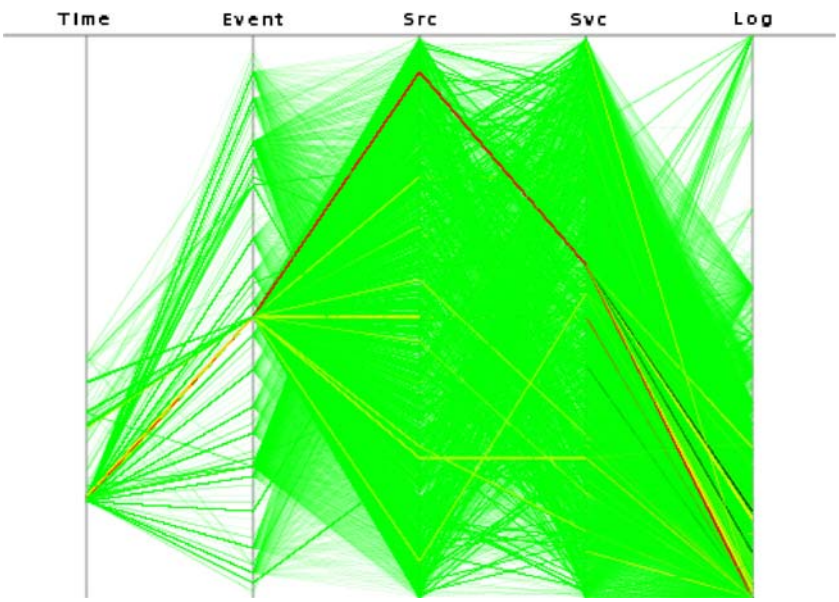
10.2 Dissecting the eventlogs file

The **eventlog** file looks like:

```
2008-09-18 04:04:54|2008-09-18
 04:04:54|ec_console_log| \ src:::
 c0-0c0s3n0|svc:::c0-0c0s3n0|
2008-09-18 04:04:54|2008-09-18 04:04:54|
 ec_console_log| \
 src:::c0-0c0s3n0|svc:::c0-0c0s3n0|7
 [?25l[1A[80C[10D[1;32mdone[m8[?25h
2008-09-18 04:04:54|2008-09-18
 04:04:54|ec_console_log| \
```

[9] http://cfdr.usenix.org/data.html#cray.

Fig. 42 Cray eventlogs graph




```
 src:::c0-0c0s3n0|svc:::c0-0c0s3n0|cat:
 /sys/class/net/rsip333x1/type
2008-09-18 04:04:54|2008-09-18 04:04:54|
 ec_console_log| \
  src:::c0-0c0s3n0|svc:::c0-0c0s3n0|:
  No such file or directory
```

By looking through this log file one may discern five fields:

- **2008-09-18 04:04:54|2008-09-18 04:04:54**: time value, because of the lack of knowledge of Cray logs for why there are two identical timestamps, the first is taken. The variable **timeline** should be appropriate.
- **ec_console_log**: unknown value, which seems like an enumeration of a few possible items. It looks like an event category. The variable **enum** should be appropriate.
- **src:::c0-0c0s3n0**: should be the source node where the even come from. Again, since there are a limited number of distinct resources, the variable **enum** should be appropriate.
- **svc:::c0-0c0s3n0**: should be the destination. Just like for sources, the **enum** variable should be appropriate.
- **: No such file or directory**: is the log itself. The **string** variable is appropriate and will be put as **relative** with the other strings.

Below the perl code for the generator to handle Cray events log file:

```
print "axes {\n";
print "    timeline t [label=\"Time\"];\n";
print "    enum     e [label=\"Event\"];\n";
print "    enum     s [label=\"Src\"];\n";
print "    enum     d [label=\"Svc\"];\n";
print "    string   l [label=\"Log\",
 relative=\"true\"];\n";
print "}\n";
```

10.3 Graphing

From the PGDL source generated and using the axes pair frequency rendering, a graph is created in Fig. 42.

//-coords reveals the following facts:

- **Time range**: logs were written in a very short at very specific time. Without being accurate, one can say all logs are written between about 5 a.m. and 11 a.m.
- **Coverage**: one event (the one in the middle) covers all sources.
- **Frequency**: the red lines shows that one event occurs way more than the other, that there is a source-destination triggering most of the events and a log occurs more than the others.
- **Differentiation**: Some low-frequency logs are far away from all the other logs. Interesting.

10.4 Filtering

As it is usually interesting to start with those events that appear in low frequency on top of the fifth axis, a filtering is applied:

```
pcv -Tpngcairo event.pcv 'show
 plot > 90% on axis 5' -a
```

Which gives the Fig. 43.

No need for frequency analysis here, since it was done in the first graph. This graph outlines that a very few logs were written, that they all come from the same class of event (second axis) and a few machines among all that were in the first graph make them.

Logs generating those lines are:

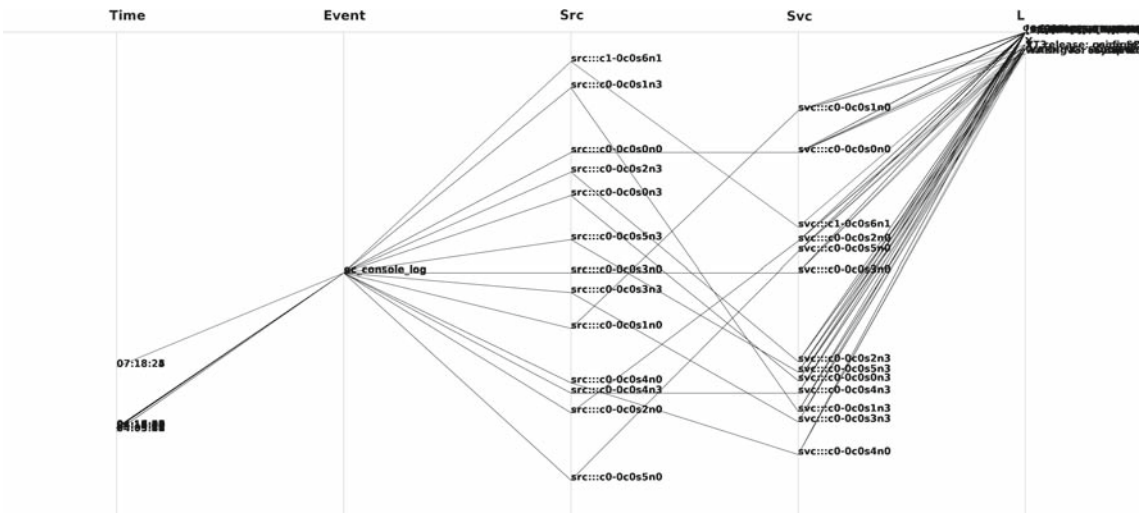


Fig. 43 Cray filtered eventlogs graph

10.4.1 Log #1

```
Bootdata ok (command line is earlyprintk
 =rca10 load_ramdisk=1
ramdisk_size=80000 console=ttyL0
 bootnodeip=192.168.0.1 bootproto=ssip
bootpath=/rr/current rootfs=nfs-shared
 root=/dev/sda1 pci=lastbus=3
oops=panic elevator=noop xtrel=2.1.33HD)
```

Interesting keywords: **earlyprintk**, **oops=panic**.

10.4.2 Log #2

```
Bootdata ok (command line is earlyprintk=
 rca10 load_ramdisk=1 CMD LINE
[earlyprintk=rca10 load_ramdisk=1
 ramdisk_size=80000 console=ttyL0
bootnodeip=192.168.0.1 bootproto=ssip
 bootpath=/rr/current
rootfs=nfs-shared root=/dev/sda1
 pci=lastbus=3 oops=panic elevator=noop
xtrel=2.1.33HD]
```

Interesting keywords: **earlyprintk** x2, **CMD LINE**, **oops=panic**.

10.4.3 Log #3

```
Bootdata ok (command line is earlyprintk=
 rca10 load_ramdisk=1 Kernel
command line: earlyprintk=rca10
 load_ramdisk=1 ramdisk_size=80000
console=ttyL0 bootnodeip=192.168.0.1
 bootproto=ssip bootpath=/rr/current
rootfs=nfs-shared root=/dev/sda1
 pci=lastbus=3 oops=panic elevator=noop
xtrel=2.1.33HD
```

Interesting keywords: **earlyprintk** x2, **Kernel command line**, **oops=panic**.

10.4.4 Log #4

```
Lustre: garIBlus-OST0005-osc-
 ffff8103f3fab800: Connection to service
garIBlus-OST0005 via nid 19@ptl was
 lost; in progress operations using this
service will wait for recovery
 to complete.
```

Interesting keywords: **Connection to service … was lost**, **wait for recovery to complete**.

As one can see, Picviz successfully help to trace events on the most used node based on both frequency analysis and high values on the log axis. Because printk are the Linux kernel print function handling anything the kernel wants to print in usual abnormal situations, it is worth taking a look at it.

```
pcv -Tpngcairo -Wpcre event.pcv 'show
 value = ".*printk.*" on axis 5 -a
```

Which gives the Fig. 44, which outlines on the fifth axis lines previously seen, and others, appearing at about 10% on the same axis, that were mixed with the other logs. By looking at those logs value, there is:

10.4.5 Log #1

```
printk: 64 messages suppressed
```

10.4.6 Log #2

```
printk: 336 messages suppressed
```

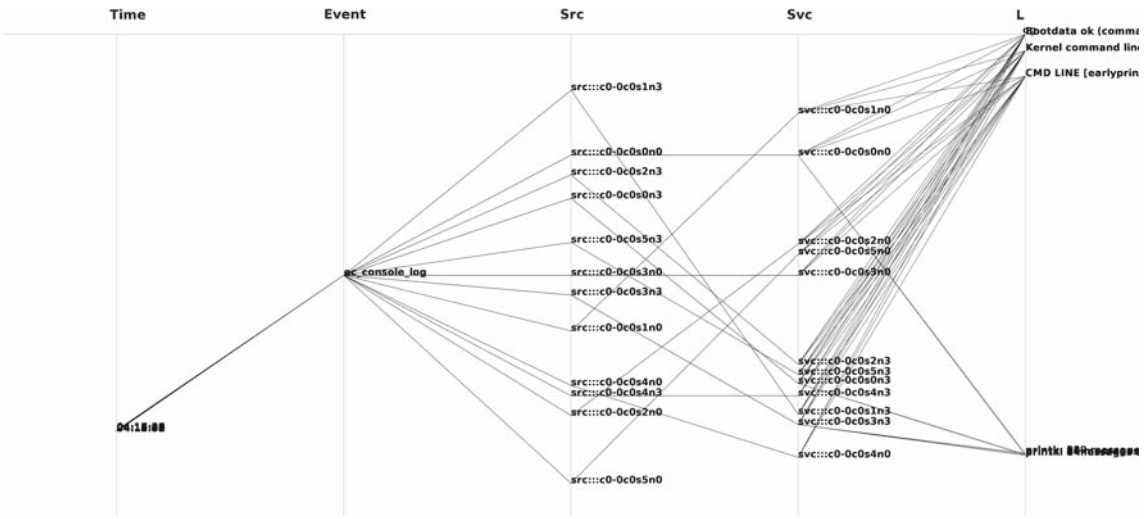


Fig. 44 Cray eventlogs graph with printk

Which reveals a set of repeated problems.

Of course, knowing exactly what occurred would require more work and knowledge of Cray systems. However using //-coords helped to understand easily what was happening on those machines, the way they log, the number of sources and what seems to be destinations.

In this section, the way //-coords were used may be looked as if one would have use the **grep** tool. Actually, the first generated picture and its frequency analysis helped to target data that could be most likely the source of an issue. Then, because lines were drawn far away from all the other and in low frequency, it went fairly easy to spot and focus on the **printk**.

11 Botnet analysis

Botnets attacks consists of spreading a malware to usually vulnerable Windows systems to get a maximum of machines. This big amount of machines allows an attacker to spread spams and make denial of services (DoS) attacks. Botnets can be controlled from IRC commands or similar custom protocol. Also files can be sent through various nodes using a peer to peer protocol [5].

When facing a Botnet attack, one will see connections from a big load of different source IP addresses. Making it hard to block. Most of the time, empirical tricks are being used to kill an ongoing DoS attack, such as renaming the target web file, etc.

This section does not intend to explain details of a botnet attack, but how such an attack looks like and how it can be understood to help defending against it.

Because Picviz is good with such a number of events, a Botnet attack was captured and Argus was used to clear out network flows, it was used to generate a //-coords graph.

In this example, two types of images based on the same data were generated. Only the color varies:

- Figure 45: UDP traffic is in blue, TCP traffic in yellow
- Figure 46: Frequency analysis per pair-axes wise

Based on those two figures, it is easy to outline:

- The attack happens on two times: the first starting at about 1 p.m. and stopping 30 min after. The second attack is bigger: starting at about 2 p.m. and stopping 3 h latter.
- The TCP flow starts during the second attack and last during a short period of time.
- The frequency analysis is more interesting: one can easily see that several IP addresses knocked the same source port. Also, one single source IP receives most of the traffic. This is because it is the target machine.
- When superposing the two pictures, most of the generated traffic happens at the same time TCP traffic happens.

As an emergency counter-measure, two solutions can be considered: blocking any UDP traffic that is not DNS requests, NTP traffic or any critical service required by the network. Also, since the source port is shared for most of IP sources, it seems this value was hardcoded. Blocking this specific port may help resolving the problem for the time the administrator can take to understand more about the attack.

In all, even though the botnet understanding requires some time, using //-coords to see obvious facts can help having more time to investigate deeper.

12 Apache access.log analysis

Trying to find evil requests within 2 years, the first thing one may do is to use known-patterns against the log. Picviz allow-

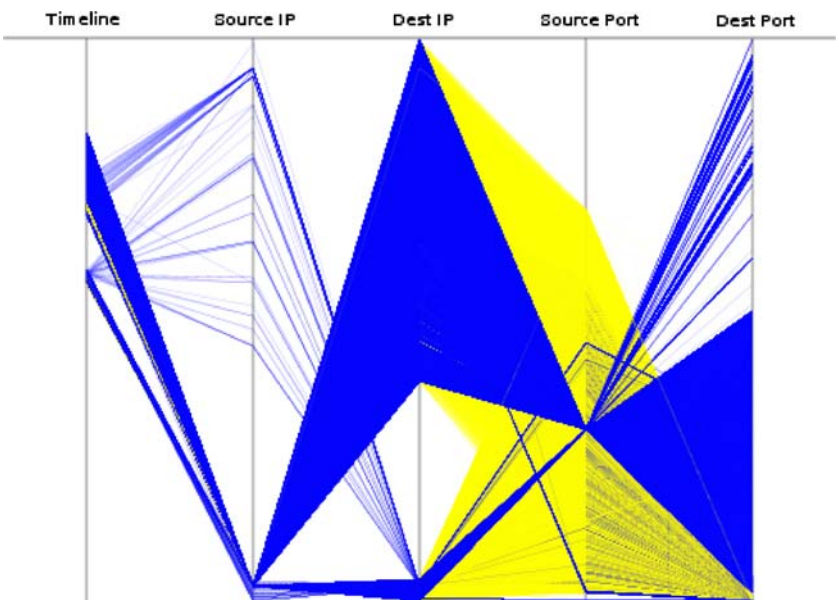


Fig. 45 TCP versus UDP botnet analysis

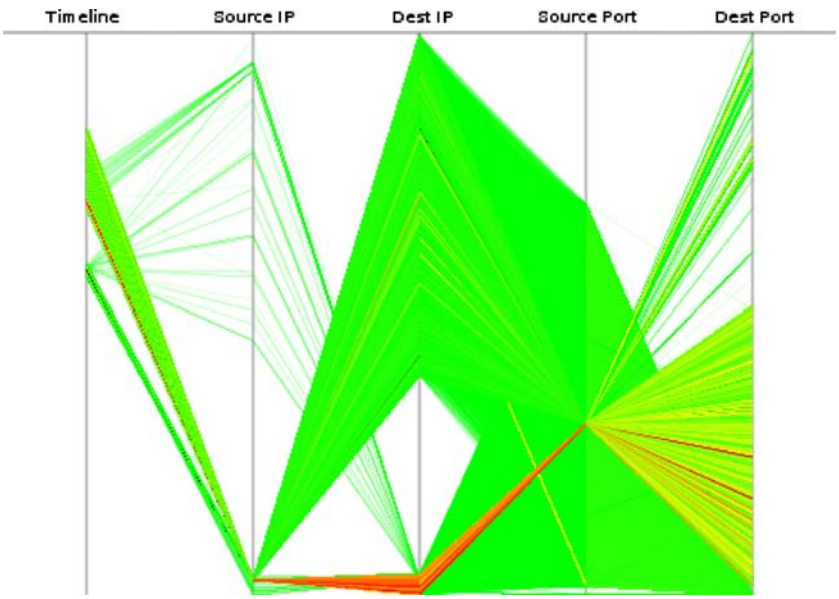


Fig. 46 Botnet frequency analysis

ing one to search for what she/he was not even looking for, going step-by-step into the attack discovery is what this section shows. The first thing to do is to generate a graph from the logs as they are (Fig. 47).

The frequency analysis mask is applied on every graph, using an axis-to-axis relationship. In 2 years, only two HTTP protocols were being used: HTTP/1.0 and HTTP/1.1.

To clear things a little, three axes one may find important are selected: IP address, User Agent and Request. The result of this selection can be seen in Fig. 48. What is made obvious in this graph is the red line that goes from a very specific IP address to a User Agent, and the same User Agent performing the **GET** request. This is actually the Google bot.

There are also interesting HTTP requests made, not the common GET. Looking at the activity made on the other requests is also interesting.

In Fig. 49, the same axes selection is performed, but the **GET** request type is removed. The following line was used to generate this image:

```
$ pcv -Tpngcairo -Rheatline access
 .log.pgdl -r 'value != "GET" on axis 3'
```

Putting the request made next to the IP address shows an interesting pattern (Fig. 50). All lines going from **1** to **2** look like request being made from the same machine. The way strings are placed place points on the axis according to the

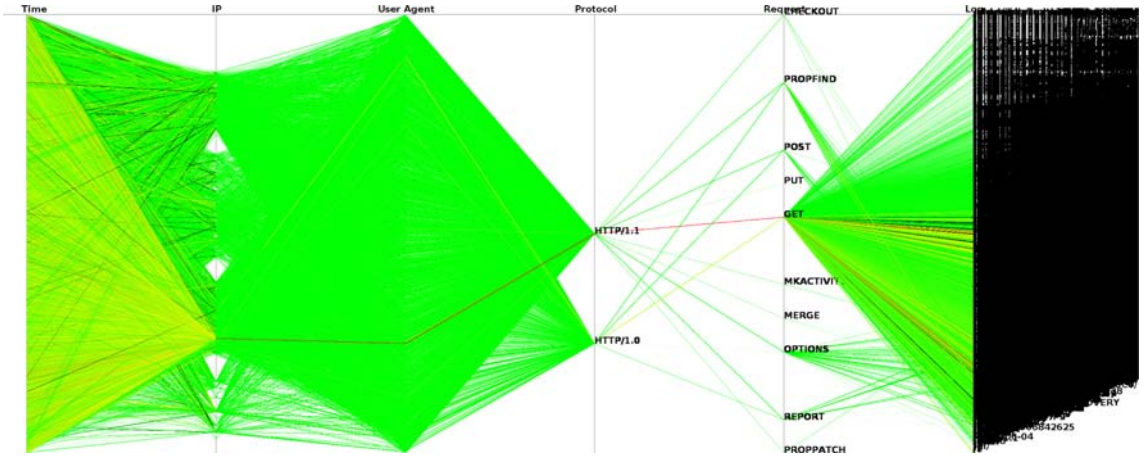


Fig. 47 First full access.log

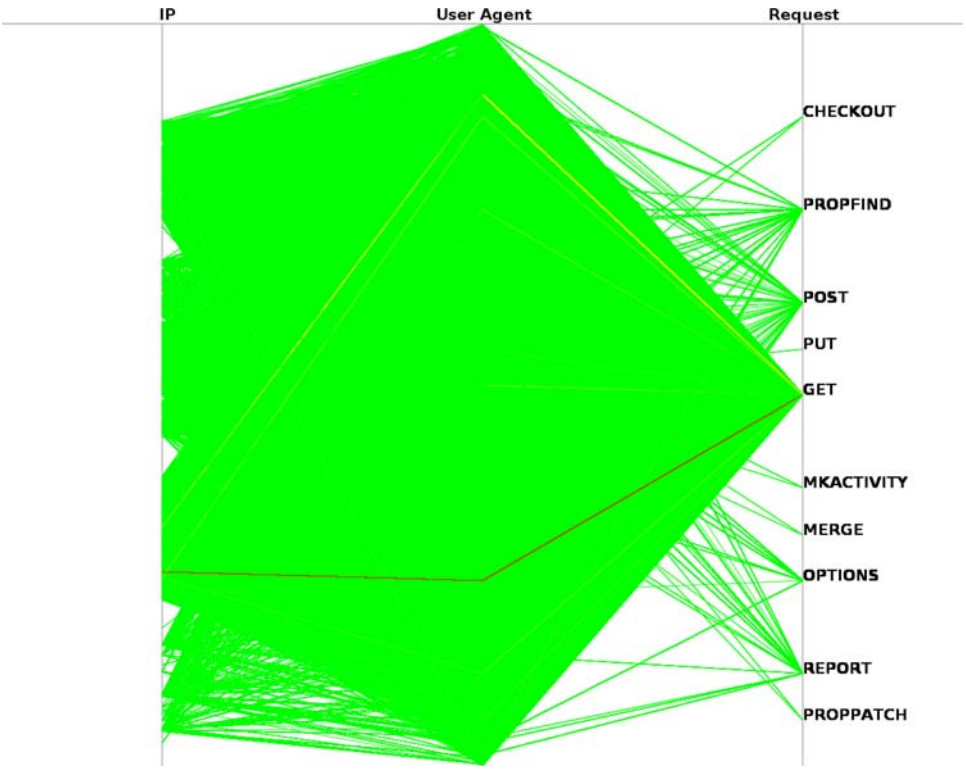


Fig. 48 Axes selection in access.log

string length. This shows requests that are bigger and bigger. Indeed, this is a Google bot fetching any available pages. The red line shows that this is event occurs frequently, show how a bot can be active (Fig. 51).

Picviz can also filter on events frequency, this allows sort events regarding how frequent they appear (filtering with **'freq < 0.0005'**).

13 WLAN images

Current technology involves Wireless when connecting to the internet. While this paper does not intend to explain WEP cracking such as in [11], the two images in Figs. 52 and 53 show some obvious patterns: one looks like normal, and the other looks like there is some sort of broadcast happening.

While understanding the attack requires how WEP cracking happens, several kind of people can deal with this problem: administrators can give the picture to analyst, who will then look for the relationships and tag the source emitting the broadcast.

What the Fig. 53 shows is the initialization phase of WEP cracking

Even though it is fairly easy to write a simple script able to detect the broadcast, the visualization and the multi-dimen-

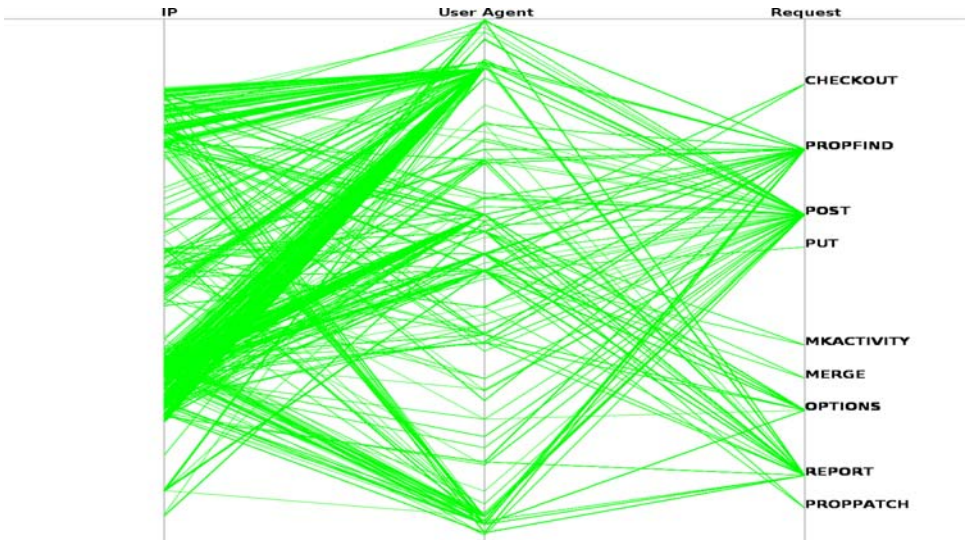


Fig. 49 Axes selection in access.log

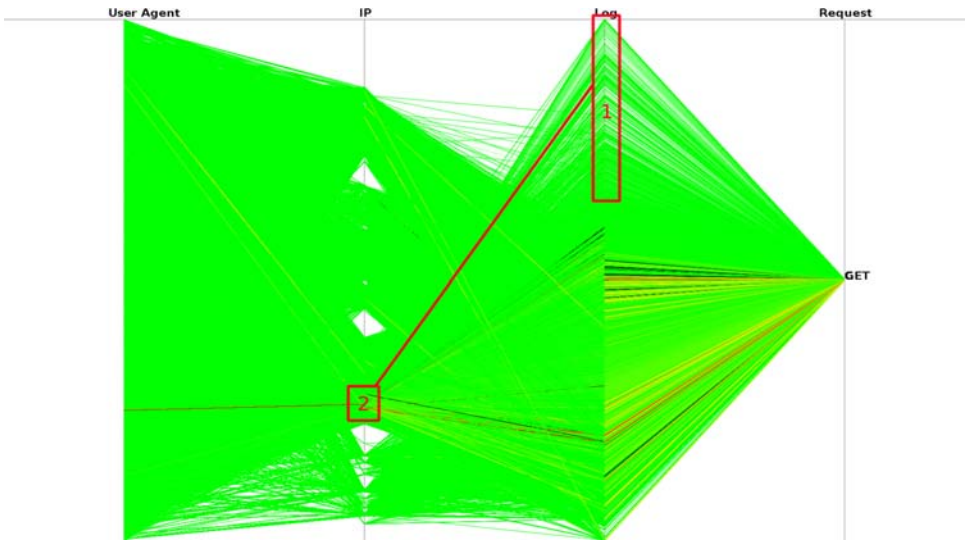


Fig. 50 Pattern in access.log

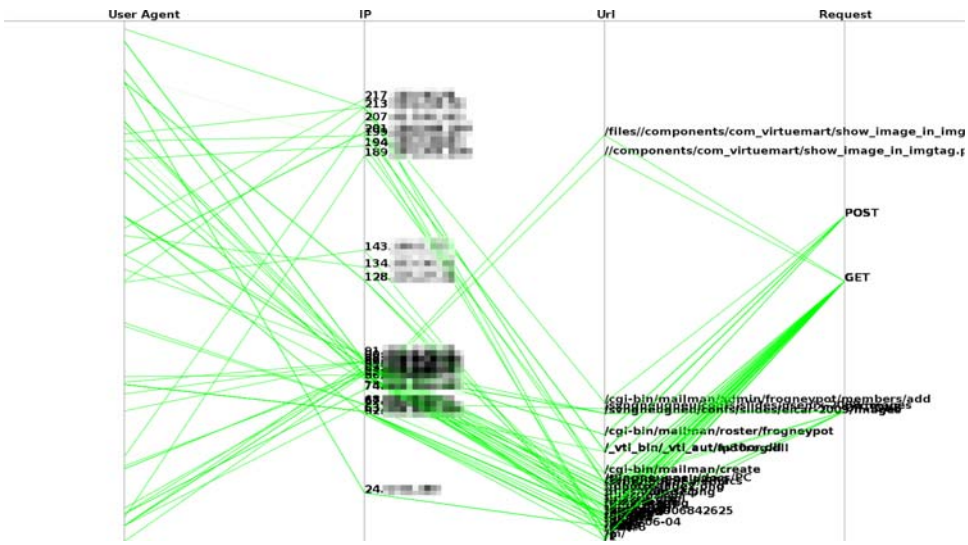


Fig. 51 Low frequency events from access.log

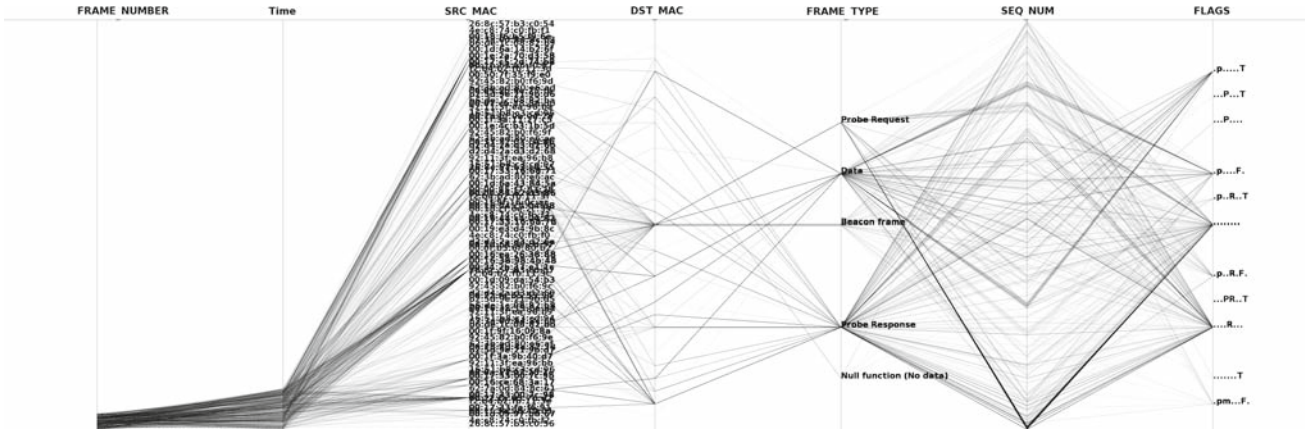


Fig. 52 WLAN without any attack

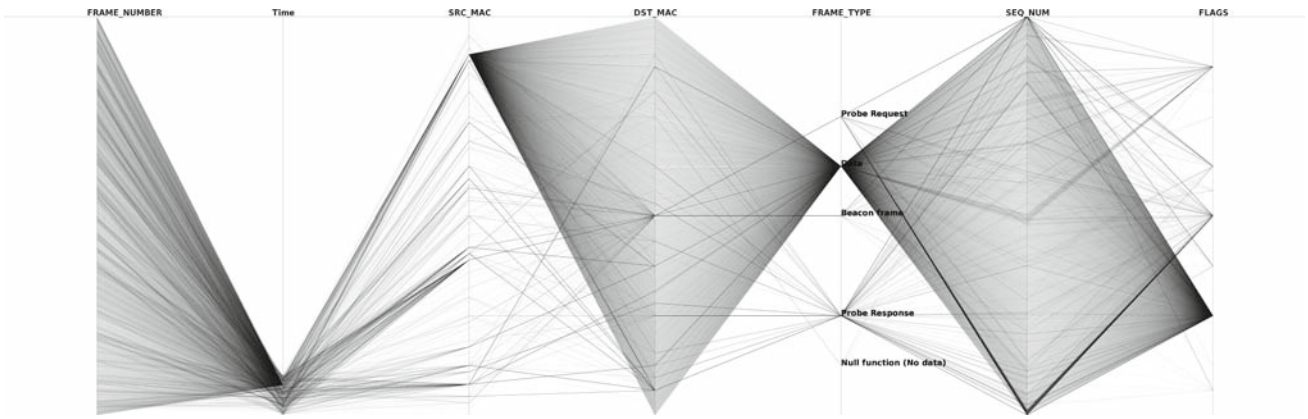


Fig. 53 WLAN initialization vector generated on the fly before a WEP key cracking

sion aspect of //-coords allows to spot what **looks** obvious, but would not have been without using this technique.

14 Conclusion

This paper explained a disruptive way of responding to computer security related events using //-coords. This was exclusively run from system logs and network traffic. It can of course be extended to other parts of a machine, such as system calls or application behavior.

However, this paper neither covered all theoretical aspects nor all practical usage of //-coordinates. Much remains to be done, proved and experienced in both directions.

First of all, on the theoretical background:

- one should decide whether //-coordinates are efficient or not in analysing **randomly** generated points of mathematically well defined object.
- one should precisely explain how and why //-coordinates can be of any interest on datasets **not coming** from any mathematical definition (such as one encounters in computer security).

On the practical background of computer security, we proved how efficient //-coordinates images can be to help security administrators to find rare and unexpected events, as much as understand more complex mechanisms hidden in huge amounts of information.

We hope to investigate soon on the two questions above and report clear explanations (more or less based on some already existing literature).

Picviz offers a very efficient tool to get a quick and faithful look into gathered datasets: frequency is taken into account (axis by axis or on all axes together), offering a transversal information correlation which proved not to be obvious.

The generated image can be very technical and has its learning curve to be able to tune and help to get an efficient image one can easily find things in. However, even when reacting as a naive person, generating a picture a day can keep the doctor away and help to discover tendencies before it is too late. The kind of images Picviz can generate can be read by several kind of people and thus improves the network security reaction at several layers.

The future of Picviz will be to automate visually correlations and go further than //-coords to make a better representation of axes correlations.

References

1. Conti, G., Abdullah, K.: Passive visual fingerprinting of network attack tools. In: VizSEC/DMSEC '04: Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security, pp. 45–54. ACM, New York, NY, USA (2004)
2. Gansner, E.R., Koutsofios, E., North, S.C., Phong Vo, K.: A technique for drawing directed graphs. IEEE Trans. Softw. Eng. **19**, 214–230 (1993)
3. Grinstein, G., Mihalisin, T., Hinterberger, H., Inselberg, A.: Visualizing multidimensional (multivariate) data and relations. In: VIS '94: Proceedings of the Conference on Visualization '94, pp. 404–409. IEEE Computer Society Press, Los Alamitos, CA, USA (1994)
4. Hamming, R.: Error detecting and error correcting codes. Bell Syst. Tech. J. **29**, 147–160 (1950)
5. Holz, T., Steiner, M., Dahl, F., Biersack, E., Freiling, F.: Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In: LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats, pp. 1–9. USENIX Association, Berkeley, CA, USA (2008)
6. Inselberg, A., Avidan, T.: The automated multidimensional detective. In: INFOVIS '99: Proceedings of the 1999 IEEE Symposium on Information Visualization, p. 112. IEEE Computer Society, Washington, DC, USA (1999)
7. Inselberg, A., Avidan, T.: Classification and visualization for high-dimensional data. In: KDD '00: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 370–374. ACM, New York, NY, USA (2000)
8. Inselberg, A., Dimsdale, B.: Parallel coordinates for visualizing multi-dimensional geometry. In: CG International '87 on Computer Graphics 1987, pp. 25–44. Springer, New York, NY, USA (1987)
9. Inselberg, A., Dimsdale, B.: Multidimensional lines ii: proximity and applications. SIAM J. Appl. Math. **54**(2), 578–596 (1994)
10. Levenshtein, V.I.: Binary codes capable of correcting deletions, insertions, and reversals. Tech. Rep. 8 (1966)
11. Stubblefield, A., Ioannidis, J., Rubin, A.D.: A key recovery attack on the 802.11b wired equivalent privacy protocol (wep). ACM Trans. Inf. Syst. Secur. **7**(2), 319–332 (2004)